Post-Silicon Validation: Automatic Characterization of RF Device Nonidealities Via Iterative Learning Experiments on Hardware

Barry Muldrey, Sabyasachi Deyati, and Abhijit Chatterjee
Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[bmuldrey3, sdeyati3]@gatech.edu, chat@ece.gatech.edu

***Abstract*—Recent studies show that increasing numbers of design bugs are escaping to post-silicon due to the complexity of advanced designs and the lack of adequate verification tools that can validate complex electrical interactions between electrical subsystems on an integrated circuit. In this paper, we present a novel tool for post-silicon validation of mixed-signal/RF circuits through cooperative test stimulus generation and behavior-learning. The implemented technique leverages iterative supervised learning techniques to comprehensively diagnose anomalies between the input-output behavior of the silicon device and the corresponding behavior predicted by its reference model. This results in both a more efficient validation test stimulus and an improved behavioral model which captures non-ideal silicon behaviors. Preliminary results on RF devices prove the feasibility of the proposed validation methodology.**

I. Introduction

In this paper we present a novel tool for post-silicon validation: simultaneous co-evolution of validation test stimuli and system models, particularly analog/RF systems. The unique benefit of this tool arises from careful integration of two competing methodologies: one seeking to expose and maximize behavioral discrepancies between the device-under-validation (DUV) and its model, and the other seeking to learn and embed new behaviors into the model so as to minimize discrepancy between the DUV and its model.

In the following section, we present our approach to concurrent evolution of the validation test stimulus and the model of the design under validation (DUV) through repeated stimulus generation and model update using behavior learning kernels. Discrepancies between the learned model and the silicon implementation of the DUV are minimized or eliminated and a suite of validation stimuli are produced. Section III details our implementation of the learning kernel within the model, namely the sparse Wiener network (SWN); its neural behavior, network behavior, and training algorithms are discussed and analyzed. Section IV presents a test case in which a hardware RF power amplifier is validated against a basic pre-silicon model using the proposed technique. Section V discusses a more elaborate test scenario in which a simulated RF transceiver is validated against its pre-silicon model using the proposed methodology. The paper concludes with a discussion of the results, potential benefits of the technique, and trajectories for new work in Section VI.

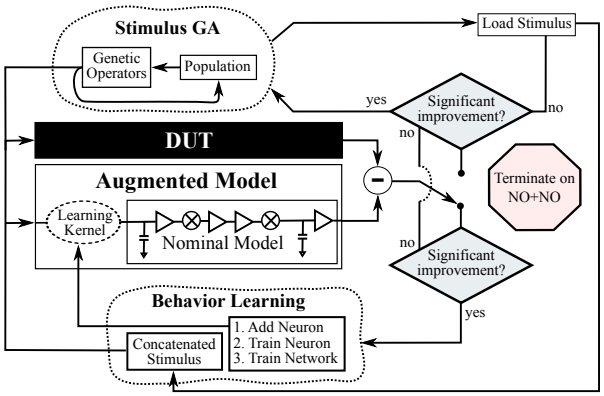

Fig. 1. System Overview

II. Co-Evolution of Validation Stimulus and Model

Post-silicon validation of mixed-signal/RF circuits using co-evolution of the test stimulus and the DUV model presents two key challenges:

(A) What test stimuli should be applied to the DUV in order to expose behavioral differences between the DUV and its implementation in silicon?
(B) How should the behavioral model of the DUV be modified or augmented to capture behavioral discrepancies between the DUV and its reference model?
(C) How should stimulus generation and model refinement be integrated?

The methodology presented here addresses both of these questions in turn.

A. Stimulus Generation

In general, pre-silicon validation techniques fail as a result of modeling and/or simulation shortcomings arising from either unknown physical interaction between subsystems/ components or from practical simulation-time limitations. Typically, assumptions about the behavior of the systems/ subsystems/ components are made to choose which combinations of subsystems/ components should be simulated, and what test stimuli should be used on each. The success of this design verification approach therefore hinges upon the accuracy of pre-

2380-6923/16 $31.00 © 2016 IEEE
DOI 10.1109/VLSID.2017.86

IEEE computer society

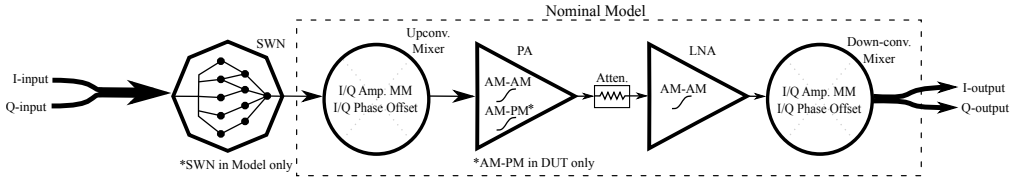

Fig. 2. Nominal Model and DUV

silicon simulation models, and additionally any tests derived from these can be rendered ineffective due to any inaccuracies [1].

We have developed a novel methodology for on-the-fly analog test stimulus generation which uses a genetic optimization algorithm (GA) [2]–[4]. It evolves an initial set of random stimuli into highly engineered stimuli which excite behavioral discrepancies between two entities: a silicon implementation of the DUV and its model. No knowledge of either the DUV or its model is required *a priori* by the algorithm; the only assumption is: if the mappings of inputs to observable outputs are equivalent, then their behavior is equivalent.

The stimulus generation algorithm simultaneously excites both the DUV and its model with candidate stimuli from a population of stimuli and captures the system's response to each. This is repeated while a tailored GA evolves the population using frequency-domain sum-squared-error between the DUV's and its model's response as the fitness function. A quantized magnitude of the stimulus in the frequency domain over a finite bandwidth is used as the genetic representation of the solution stimulus. In this work, the concepts of fitness and the genetic representation have been adapted so as to simultaneously optimize stimuli for a two-input, two-output system (Fig.2).

As implemented, the algorithm will converge on stimuli which cause the greatest (in magnitude) net discrepancy between the DUV and its model. These discrepancies can be due to basic parametric differences between the two, and as such they can be accounted for through parametric tuning [5]–[8]. Post-silicon validation however, seeks to expose anomalies *beyond* parametric variation. The proposed stimulus generation approach succeeds in that it has a "heat-seeking" effect in identifying and illuminating individual behavioral discrepancies; however, difficulties arise from the potential for one discrepancy to mask others. For example, consider an amplifier whose model suggests it should have a gain of 10 and a fabricated device whose gain is 7. In this case, the algorithm produces an approximation of a square wave which maximizes operation in areas of the input space with the largest discrepancy; any non-linearity present would be masked by the overwhelming effect the linear gain mismatch has on the optimization.

To address this, we present a key feature of this work: test generation is performed in an iterative manner over several passes; in each pass, a dominant behavioral discrepancy between the DUV and its model is excited, the discrepancy is then integrated into the model, test generation is performed again to reveal additional DUV behaviors potentially masked in earlier test generation passes.

B. Model Augmentation

Behavioral models are created and parameterized based on expectations, and in general, their functionality can vary across at least as many dimensions as there are parameters. However, one can easily imagine a case in which a DUV exhibits behaviors which its model lacks: a simple linear amplifier model might have linear gain, slew-rate, and gain-bandwidth product parameters, yet any implementation will exhibit some amount of distortion when stressed. There is no combination of values of the provided parameters which can model the non-linearity causing the distortion. This is a fundamental trade-off of modeling, and it is for this reason that a highly flexible, highly adaptable, and easily trainable behavior-learning element is required. And so the idea of the behavior-learning kernel is to provide a means of selectively imparting additional degrees of freedom to parametric behavioral models.

Neural networks are among a class of tools that lend themselves to behavior-learning, fundamentally a function approximation problem. We began our work using low-order radial-basis networks (RBN), but as behavior complexity increased, we have come to use a variant of the RBN, described in Section III. To aid in designing effective training algorithms, care must be taken to design the learning kernels so that their component and parametric contributions to system level behavioral change can be measured in a meaningful way; that is, the complexity of the kernel cannot be so great that its parameters lose contextual meaning (as can be the case in deep and many-layered neural networks).

C. Stimulus Generation and Model Augmentation in Concert

A stimulus generation algorithm similar to [2] has been introduced which relies only on a DUV and its model, and a model augmentation method has been introduced which relies only upon a DUV and its test stimulus. Assuming that the DUV's behavior is fixed in both cases, one realizes that the processes of stimulus generation and model augmentation are co-dependent. We propose that these processes should be executed in parallel as depicted in Figure 1. As implemented, the stimulus generation algorithm executes until improvement rate drops below a threshold; the resulting stimulus is added to a set of training stimuli; the augmented model is trained over the set until its performance improvement falls below a threshold

rate; and then the process repeats. Through this co-evolution technique, one gains valuable insight into the relative complexities of the discovered behaviors and measurable knowledge of the contributions of individual stimuli in exercising the system. Most of all, the co-evolution gives confidence and measureability to the notion of validity, neither of which would come by simple serial execution of stimulus generation and model augmentation tasks. A key outcome of this approach is that the learning kernels within the augmented model now encapsulate all the stimulated differences between the DUV and its original (un-augmented) model. Such information can be leveraged to manually modify the DUV's simulation model to exhibit more realistic post-silicon behaviors or debug the design to ensure that DUV behaviors observed in silicon match the intended behavior of the DUV as dictated by its reference model.

Algorithm 1 System Algorithm
Input: DUV, $model$
Output: $model$

```
testStim=generateStim(DUV,model)
modelout=sim(model,testStim)
duvout=sim(DUV,testStim)
performance = sum((modelout − DUVout)^2)
WHILE(performance > thresh)
    model=trainSWN(DUV,model,stimulus)
    testStim=generateStim(DUV,model)
    modelout=sim(model,testStim)
    duvout=sim(DUV,testStim)
    performance = sum((modelout − DUVout)^2)
ENDWHILE
```

III. The Sparse Wiener Network as a Learning Kernel

The Sparse Wiener Network (SWN) as described here has two inputs: a signal input and an activation input, and has one output. In this section, a process is described by which SWNs are constructed and placed within behavioral models to give the models behavior-learning abilities which go beyond those afforded by the models' parameters. These SWN-augmented models will be herein referred to as "augmented models," and "error" will refer to the difference between the response of the DUV and model output to the applied stimulus. Individual neurons in the SWN are structurally similar to the neurons in Radial-Basis Networks (RBN) [9], with the major distinction that SWN neurons allow for neuron activation independent of the input whereas RBN neurons are activated by and process the same input. This allows for the activation of neurons to originate in other signals and to be of a dimension less than or equal to that of the signal input. Additionally, in order to exhibit memory effects (or compensate for them), delay taps can be placed at either the signal input, the activation input, or both. Note that the amount of delay at either input need not equal the delay at the other, nor is there a requirement for any delay at all; however, the training of SWNs to learn memory-effected behaviors is beyond the scope of this paper [10].

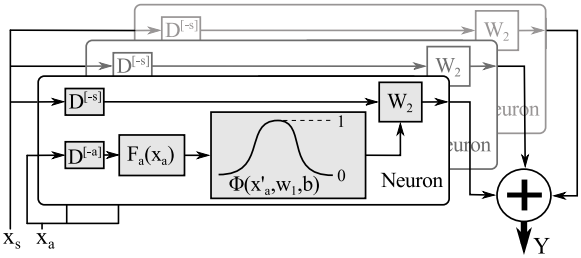

Fig. 3. A Three Neuron SWN

Each neuron receives both an N-dimensional signal input, $\vec{x_s}$, and an M-dimensional activation input, $\vec{x_a}$. The signal input, $\vec{x_s}$, is the one that the network transforms; the activation input, $\vec{x_a}$, determines the magnitude and nature of the transformation. These two inputs can be configured so that a signal determines its own degree of transformation (e.g. polynomial distortion). Each neuron also has an activation function, F_a, which operates on $\vec{x_a}$ to modify the activation signal, $\vec{x'_a}$:

$$\vec{x'_a} = F_a(\vec{x_a}) \tag{1}$$

F_a can be used to change the shape of the neural response relative to the signal space and can reduce the dimensionality of $\vec{x_a}$, easing the task of training. For example, a frequency-depended behavior can be more easily learned if signals are first translated into the frequency domain; in this case, the FFT can be used as the activation function. The dimensionality of $\vec{x'_a}$ will be notated "P".

Next, a synaptic function, $\Phi(\vec{x'_a}, w_1, b)$, is applied to the output of the activation function (w_1 and b are programmable weight and bias values respectively) in order to get an activation output, $\vec{y_a}$. In this work, the Gaussian Radial Basis Function is used:

$$\vec{y_a} = e^{-b\|w_1 - \vec{x'_a}\|} \tag{2}$$

where $0 < b$ and $w_1 \in \mathbb{R}^P$. Therefore, w_1 determines the center of the neuron's activation in the activation function range, and b determines the radius of activation in the same range (high values of b causing the neuron to activate more locally). With the neurons' activation defined, the output of the neuron, $\vec{y}$, can now be defined as a function of $\vec{y_a}$, the neuron's signal input: $\vec{x_s}$, and a programmable Wiener coefficient: w_2.

$$\vec{y} = w_2(\vec{x_s} \cdot \vec{y_a}) \tag{3}$$

where $w_2 \in \mathbb{R}^N$.

A SWN network simply requires a collection of SWN neurons as just described. As a matter of semantics, the neurons should share the same signal input and the same activation input (irrespective of individual neuron delays). The output, $\vec{Y}$, of a network of k neurons is given by the sum of each neuron's output, $\vec{y_i}$:

$$\vec{Y} = \sum_{i=1}^{k} \vec{y_i} \tag{4}$$

Figure 3 depicts an individual neuron inside an SWN.

A. Training SWN Learning Kernels within Behavioral Models

Algorithm 2 Algorithm for SWN Training
Input: DUV, $model$, $testStim$
Output: w_1,w_2, and b vectors
$modelout$=sim($model$,$testStim$)
$duvout$=sim(DUV,$testStim$)
$error$=mag($modelout - duvout$)
for n=1:$maxNumNeurons$ **do**
  $w_{1,n}, w_{2,n}, b_n$ = addNeuron()
  ROI=find($error$==max($error_vec$))
  for i = len($actFnList$) **do**
    $actin$=fwdTrace($model$,$testStim$)
    $actout$=$actFun_i$($actin$)
    $w_{2,n,i}$=newtonMin($error$,$actout(ROI)$,$w_{2,n,i}$)
    $b_{n,i}$ = newtonMin($error$,$actout$,$b_{n,i}$)
    $w_{1,n,i}, w_{2,n,i}, b_{n,i}$= newtonMin($error$,$actout$,$w_{1,n,i}, w_{2,n,i}, b_{n,i}$)
    $performance_i$ = sum($(modelout - duvout)^2$)
  end for
  implement(index($performance$==min($performance$)))
end for

In order for a behavioral model to leverage the adaptability of the SWN, the two must be integrated into an augmented model. The SWN can be placed in series or in parallel with any combination of modules within the behavioral model, with its activation input coming from anywhere within the system; configurations utilizing feedback can also be realized; however, it is beyond the scope of this paper to address the relative merits and difficulties in placing and training SWNs in various locations and configurations within a model. Choice of SWN kernel placement is briefly addressed in Section V-B

Traditional network training algorithms require three things: knowledge of the network's input signal, knowledge of the target SWN output signal, and differentiable neuron activation functions. Because SWNs in this work are used to learn the behavior of silicon DUVs whose internal signals are not observable, the SWN's target output is not known; therefore traditional training algorithms cannot be used. A training method is now presented which requires knowledge of only *system-level* target output.

To make training of the network more tractable, each neuron's synaptic function should have finite, practically finite, or semi-infinite support in the activation function range. That is, the neuron should activate only locally to its center (though technically having infinite support, the Gaussian RBF that is used in this work has a practically finite support which is controlled by b). Thus, it is assumed that if a network's neurons are added and trained individually to some locally optimal state, then the network on the whole will be near a local optimum as well. This method does not propose to find a globally optimal solution.

Briefly, the methodology for training embedded SWNs to capture behaviors unexplained by a parametric model is explained in Algorithm 2 (all optimizations are performed to minimize errors in a least squares sense).

IV. RF Power Amplifier Validation

An RF transmitter system similar to that depicted in Fig.2 was set-up in hardware using Maxim Semiconductor MAX 2039 up- and down-converting mixers with a Maxim MAX2242 power amplifier in the RF portion of the chain. The system model was initialized using the manufacturer's supplied conversion gain for the mixers and the PA's nominal linear gain. Iterative stimulus generation was performed and error measurements were taken throughout (Fig.4). In total, 8 stimuli were found and 34 neurons were added and trained within the model. Figure 5 presents the pre-silicon assumption of behavior, the DUV behavior, and the augmented model's behavior, and Fig.6 shows the concatenated stimuli after the 8 iterations. It is important to note the SWN's ability to capture the hysteretic behavior of the amplifier.

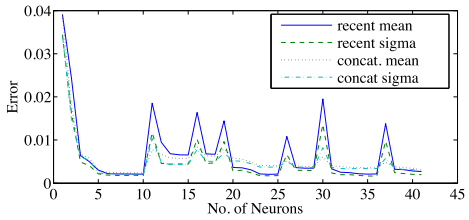

Fig. 4. PA error over iterations

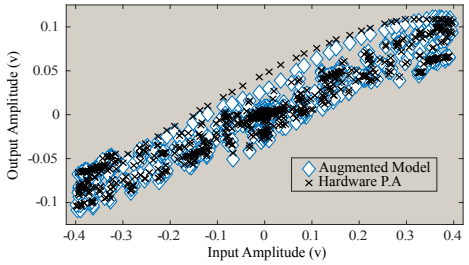

Fig. 5. PA model performance

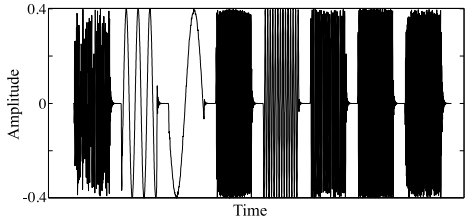

Fig. 6. PA concatenated stimuli

V. Full Transceiver Validation

In this section, we present a test case involving a full transceiver chain. The model includes many non-idealities: several in each component of the chain. The system in this

example is a full transceiver chain configured in "RF near-end loopback," with the output of the transmitter, externally attenuated and then fed back into the front-end of the LNA as depicted in Fig2. The DUV in this test case *is identical to the model with the exception of the presence of minor Amplitude-to-Phase (AM-PM) effects in the power amplifier (PA)* which the model does not account for.

The quadrature upconversion mixer modulates a quadrature local oscillator with the incoming I-Q baseband signals. Its model accounts for path-amplitude mismatches between the I and Q channels and a static phase offset between the in-phase and quadrature LO signals (i.e. quadrature LO phase $\neq -90$). Nonidealities included in the PA are 3rd order and 5th order amplitude nonlinearities, also known as AM-AM effects or gain compression. The LNA is modeled by a linear gain and 3rd- and 5th- order amplitude nonlinearity. Finally, the signal passes through the quadrature downconversion mixer whose model imposes some amount of path amplitude mismatch between I and Q channels and imparts some amount of static phase offset between the in-phase and quadrature-phase downconverting LOs.

An instantiation of this system was chosen to serve as a model with parameters presented in Table I. These parameters are assumed to be nominal with respect to process variation.

A. The DUV – Virtual Silicon

For the purposes of this experiment, the DUV is simply an instance of the nominal model, but whose PA has been modified to exhibit AM-PM effects through use of a 3rd order polynominal to map the incoming signal's amplitude to degrees of phase suppression:

$$\delta\theta(|v(t)|) = \beta_1|v(t)| + \beta_2|v(t)|^2 + \beta_3|v(t)|^3 \qquad (5)$$

Values of β used in the DUV are: $\beta_1 = -0.05$, $\beta_2 = 0$, and $\beta_3 = -4.89$. It is observed that while the difference in behavior is limited to some AM-PM effects within the PA, the systems' behaviors vary much more intricately than the simple AM-PM behavior would suggest.

TABLE I
NOMINAL MODEL PARAMETERS

Component	Param. Desc.	Param. Value	Unit
Upconv. Mixer	I-gain	0.985	v/v
	Q-gain	1.03	v/v
	I-phase $\delta\theta$	0.0524	rad
	Q-phase $\delta\theta$	-0.0698	rad
PA	linear gain	31.6	v/v
	3rd order	-59.598	v/v^3
	5th order	-53.7200	v/v^5
LNA	linear gain	5.6	v/v
	3rd order	-31.7901	v/v^3
	5th order	-17.6999	v/v^5
Dnconv. Mixer	I-gain	0.99	v/v
	Q-gain	0.975	v/v
	I-phase $\delta\theta$	0.02	rad
	Q-phase $\delta\theta$	0.03	rad

B. Application of the Proposed Method

Stimulus is applied to both the DUV and its model, and the difference of the output signals obtained is analyzed. The magnitude of the difference signal, $\bar{\epsilon}$, is herein referred to as the "error signal." At this point, a SWN is inserted at the baseband input of the model as depicted in 2. The augmented model's input is both the SNW's signal input and its activation input, and the SWN's output now provides the input for the nominal model.

Because the SWN precedes the nominal model in series, it must have an initial state which does not effect the performance of the model; it must pass its input undisturbed to its output. In order to achieve this, the first neuron is created and given a bias value $\beta = 0$ and a coefficient, $w_2 = 1$. This neuron can now be trained; we have chosen to introduce a second neuron before training the network.

An additional SWN neuron is now added. The algorithm implemented first attempts to center the neuron on the input point which causes the greatest error at the output and chooses the first activation function from the list provided.The new neuron training procedure described in Section III-A above, is followed with very coarse optimization tolerances in the interest of speed. This same procedure is attempted for all activation functions provided to the SWN, and the activation function providing the greatest magnitude of improvement (greatest decrease in DUV/augmented-model disagreement) is adopted permanently. The new neuron is trained again according to the same procedure, but with stricter optimization tolerances. After the new neuron is placed and trained, the entire network can be trained.

The performance increase through training the neuron is measured as the ratio, P, of the new error to the old; if P falls below a threshold (0.99 in this work) an additional neuron is added, a new target point is chosen, the new neuron and network are trained, and the performance increase ratio is taken once again. However, if the ratio exceeds the threshold, the current stimulus is deemed to have been exhausted, and generation is again performed, but upon the recently improved augmented model. It is important to note that if the entire network is being trained at any step throughout this process, neurons which had been trained on prior data can potentially "unlearn" old behaviors if the old training data isn't presented with the new. To address this, we simply concatenate stimuli successively to the previous iteration's stimulus.

C. Results

A first attempt at augmenting the model had the SWN placed at the nominal model's output. However, the SWN was not able to achieve the expected performance increases. Due to the non-invertibility of some behaviors (e.g. gain compression) occurring within the model that we see some of the output data points with the greatest error appearing amongst relatively well-performing data points. We cannot train the SWN to distinguish between the two. For this reason, the SWN was moved to serially precede the nominal model.

The procedure described in Section II-C was iterated 6 times and a total of 18 neurons were added. Fig. 7 depicts bare model performance across the entire stimulus space, while Fig. 8 depicts the augmented model's performance across the entire stimulus space. Fig. 9 plots the mean and standard deviation in error after addition of each neuron, and Table II presents selected data.

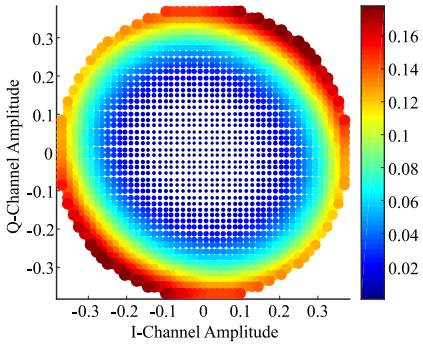

Fig. 7. Uniform sampling of stimulus space with the original model's output error corresponding to color and dot pitch

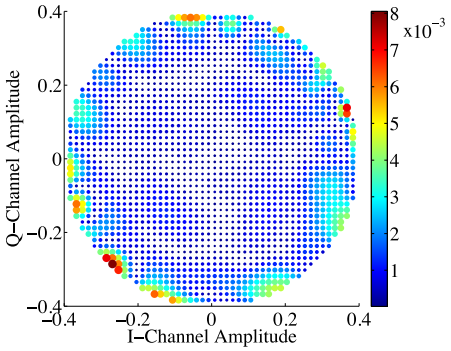

Fig. 8. Uniform sampling of stimulus space with the *augmented* model's output error corresponding to color and dot pitch

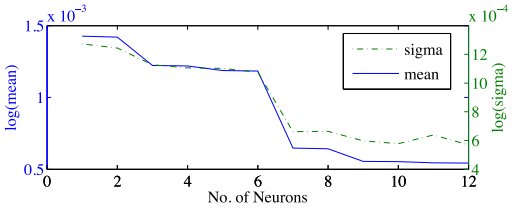

Fig. 9. Mean error and error standard deviation as neurons are added to the SWN

VI. Conclusion

In this paper we have presented a novel approach for automatically detecting and modeling design bugs in RF systems using test generation experiments on fabricated silicon. To the best of our knowledge, this is the first methodology of its kind that delivers design bug models to the designer and validation engineer for rapid design debug. The method has been validated through hardware and simulation experiments, and shown excellent results. We are currently applying the methodology to different classes of circuits and experimenting with different kinds of behavior learning algorithms to determine what gives the best results.

TABLE II
Selected Results

Feature	Value	Unit
nominal model mean error	20.7	mV
augmented model mean error	542	μV
nominal model error sigma	35.6	mV
augmented model error sigma	570	μV
neurons implemented	36	–
unique activation functions employed	7	–

Acknowledgment

The authors would like to thank the National Science Foundation (Grants ECCS:1407542 and SaTC:1441754), the Semiconductor Research Corporation (GRC Tasks 2555.001 and 2712.005), and Intel Corp. for supporting this research.

References

[1] M. Slamani and B. Kaminska, "Multifrequency analysis of faults in analog circuits," *Design Test of Computers, IEEE*, vol. 12, no. 2, pp. 70 –80, summer 1995.

[2] B. Muldrey, S. Deyati, M. Giardino, and A. Chatterjee, "Ravage: Post-silicon validation of mixed signal systems using genetic stimulus evolution and model tuning," in *VLSI Test Symposium (VTS), 2013 IEEE 31st*, April 2013, pp. 1–6.

[3] Y. Shirong, "Optimization of frequency testing stimulus based on improved genetic algorithm," in *Information Management, Innovation Management and Industrial Engineering, 2009 International Conference on*, vol. 2, Dec 2009, pp. 341–344.

[4] W. Jiawen, L. Zhigui, W. Suliang, L. Yang, L. Yufei, and Y. Hao, "Coverage-directed stimulus generation using a genetic algorithm," in *SoC Design Conference (ISOCC), 2013 International*, Nov 2013, pp. 298–301.

[5] S. Cherubal and A. Chatterjee, "Test generation based diagnosis of device parameters for analog circuits," in *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 2001, pp. 596 –602.

[6] A. Banerjee, V. Natarajan, S. Sen, A. Chatterjee, G. Srinivasan, and S. Bhattacharya, "Optimized multitone test stimulus driven diagnosis of rf transceivers using model parameter estimation," in *VLSI Design (VLSI Design), 2011 24th International Conference on*, Jan 2011, pp. 274–279.

[7] A. Banerjee, S. Sen, S. Devarakond, and A. Chatterjee, "Automatic test stimulus generation for accurate diagnosis of rf systems using transient response signatures," in *VLSI Test Symposium (VTS), 2011 IEEE 29th*, may 2011, pp. 58 –63.

[8] P. Variyam, S. Cherubal, and A. Chatterjee, "Prediction of analog performance parameters using fast transient testing," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 349 –361, mar 2002.

[9] N. Karayiannis and M. Randolph-Gips, "On the construction and training of reformulated radial basis function neural networks," *Neural Networks, IEEE Transactions on*, vol. 14, no. 4, pp. 835–846, July 2003.

[10] A. Dingankar, "The unreasonable effectiveness of neural network approximation," *Automatic Control, IEEE Transactions on*, vol. 44, no. 11, pp. 2043–2044, Nov 1999.