Fast-Accurate Non-Polynomial Metamodeling for nano-CMOS PLL Design Optimization

Oleg Garitselov[1], Saraju P. Mohanty[2], and Elias Kougianos[3]
Nano-Systems Design Laboratory (http://nsdl.cse.unt.edu)[1,2,3]
Department of Computer Science and Engineering [1,2]
Department of Engineering Technology [3]
University of North Texas, Denton, USA.[1,2,3]
Email-ID: omg0006@unt.edu[1], saraju.mohanty@unt.edu[2], and eliask@unt.edu[3]

***Abstract*—At the nanoscale domain, the simulation, design, and optimization time of the circuits have increased significantly due to high-integration density, increasing technology constraints, and complex device models. This necessitates fast design space exploration techniques to meet the shorter time to market driven by consumer electronics. This paper presents non-polynomial metamodels (surrogate models) using neural networks to reduce the design optimization time of complex nano-CMOS circuit with no sacrifice on accuracy. The physical design aware neural networks are trained and used as metamodels to predict frequency, locking time, and power of a PLL circuit. Different architectures for neural networks are compared with traditional polynomial functions that have been generated for the same circuit characteristics. Thorough experimental results show that only 100 sample points are sufficient for neural networks to predict the output of circuits with 21 design parameters within 3% accuracy, which improves the accuracy by 56% over polynomial metamodels. The generated metamodels are used to perform optimization of the PLL using a bee colony algorithm. It is observed that the non-polynomial (using neural networks) metamodels achieve more accurate results than polynomial metamodels in shorter optimization time.**

***Index Terms*—Metamodeling, Neural Networks, Nano-CMOS, PLL, Polynomial, Modeling, Circuit Optimization**

I. Introduction

The design constraints on the designer with competitive time to market discourages the use of slow exhaustive design space exploration to reach fully optimal performance. At the same time accurate circuit level, full-blown parasitic netlist based design optimization may be intractable for current nanoscale high-density complex circuits. In many cases optimization is very sub-optimal and the design is within certain design margins. For nanoscale circuits, the simulation, design, and optimization time has increased significantly due to high-integration density, increasing technology constraints, and complex device models. It is especially true for analog/mixed-signal (AMS) circuits. Following Amdahl's law, considering the slowest part in the optimization process, circuit simulations have the highest priority. Simulation is needed for AMS circuits since numerical methods cannot predict the parasitics that influence the circuit performance greatly after the physical design has been derived, hence it is hard to predict the outcome of the actual circuit. Different approaches have been proposed to either simplify the circuit (i.e. macromodeling) or predict the output values for the circuit using surrogate approaches like metamodeling, multiple regression techniques, and Design of Experiments [1], [2].

Metamodeling based design flows are investigated as approaches to reduce design cycle time. The proposed non-polynomial metamodeling design flow speeds up design process by creating accurate metamodels and uses them for optimization. Metamodeling is used in variety of different fields to predict the values of time consuming or expensive processes [3]. Creation of the metamodel starts with sampling the design space and then using mathematical approaches to create formula(s) for output prediction. For circuits, the sampling is performed using circuit simulations. There are different approaches to create the predictive formula(s). Polynomial least square regression is the most common and very widely used [1]. Its simplicity is very attractive, but it is not efficient for very high dimensional circuits (many parameters) due to the number of coefficients which is limited by memory space. To improve polynomial regression models, splines can be used. But they also have the same limitations as regular polynomials for high dimensional datasets.

Neural networks may be an answer for creating very high dimensional models [4]. For a limited amount of simulations, the trained neural network preforms almost equally well for any number of parameters. Multi-layer networks are trained in parallel for every input by adjusting corresponding weights for non-linear and linear functions. Once the network learns and conducts final adjustments for weights of the internal functions, it is able to predict the values with only the number of parameters times the number of layers functions in the model. This makes neural networks very robust. Finding the right architecture for a neural network usually requires some experimentation. A few techniques are considered in this work: different non-linear functions and varying the amount of neurons in the hidden layer. The data to create the neural networks are directly generated by SPICE. Once the neural network is generated it can predict the output value very fast, due to its small complexity. Hence, the prediction is much

[0]This research is supported in part by SRC award P10883 and NSF awards CNS-0854182 and DUE-0942629.

faster than SPICE. A trained neural network can also be used by constrained single and multi objective optimization algorithms, just like regular metamodels.

The rest of the paper is organized as follows: Section II discusses related previous research. A brief overview of the PLL circuit that was used in this paper is presented in Section III. Section IV describes the creation and use of neural networks for the PLL circuit. Experimental results are presented in Section V. The paper is concluded with directions for future research in Section VI.

II. Related Research and Contributions

The current literature is rich in research trying to speedup the design process of complex AMS circuits. Design space exploration approaches from high level descriptions of analog circuits are given in [5]. Posynomial modeling for gate sizing is presented in [6]. A layout-aware modeling approach for analog synthesis is given in [7]. A single manual design iteration design flow is proposed in [8] for fast design optimization of VCOs.

The following are selected research works that have applied neural networks in VLSI design. In [9], the author shows that neural networks can be used for circuit analysis. In [10], the authors introduce the creation of neural networks for estimating the output of operational amplifiers from a high level perspective which does not account for parasitics. In [11], optimal and Hopfield binary neural networks are used for testing stuck at fault and delay faults in digital circuits. In [12], neural networks are trained on multi-dimensional mapping between geometrical variables and the values of independent circuit elements to predict of electromagnetic behavior of vias. In [13], the authors propose to speed up simulations by replacing repeated simulation data such as polynomial and look up models with well trained neural networks. In [14], a Hopfield neural network model is used to represent digital circuit behavior. In [15], a feed-forward dynamic neural network model is developed for amplifier and mixer circuits directly from input-output large-signal measurements, without having to rely on internal details of the circuit. In [16], neural networks are used for electromagnetic susceptibility analysis and optimization of electronic devices.

The **novel contributions of this paper** are as follows. A non-polynomial metamodel based design optimization flow for analog/mixed-signal circuits is presented. For non-polynomial metamodeling different architectures of neural networks are considered to perform tradeoff analysis between speed and accuracy. As a practical demonstration of the use of the non-polynomial neural network metamodels, a physical design optimization of a 180 nm CMOS PLL is undertaken. A biologically inspired tool, the bee colony algorithm is used for optimization of the PLL physical design that uses the metamodels instead of the actual circuit (i.e. the parasitic aware netlist). It is demonstrated that the non-polynomial neural network metamodel assisted optimization is faster and more accurate compared to the polynomial metamodel.

III. The Case Study Circuit: A 180nm CMOS PLL

The phase locked loop (PLL) is a closed feedback loop system which is implemented as shown in Fig. 1. The detailed baseline design of this circuit is discussed in [17].

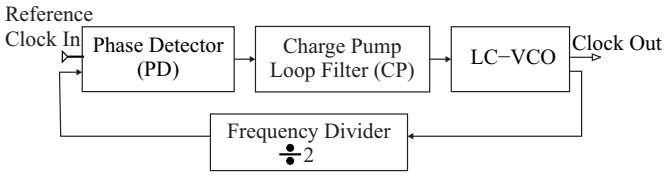

Fig. 1. Block diagram of a phase locked loop (PLL).

The physical design is shown in Fig. 2. A parasitic-aware netlist, including resistance (R), capacitance (C) and inductance (both self and mutual) (LK) is extracted from the layout. The netlist is then parameterized and used for simulations for the input data sets for each metamodel. Once the data are received from SPICE simulations, they are processed by an external tool (Matlab).

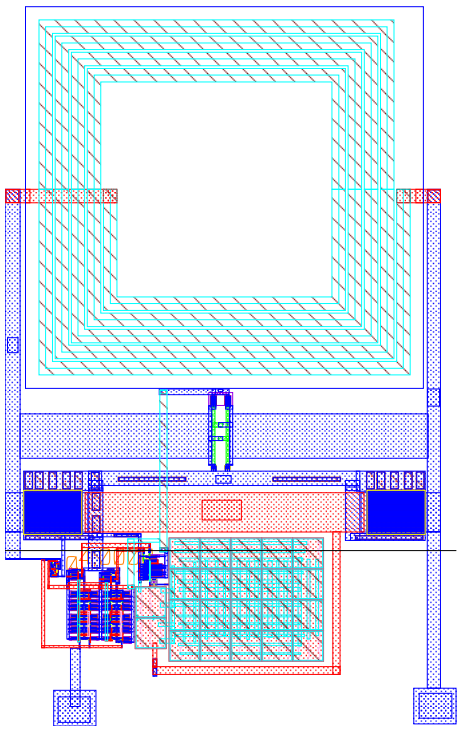

Fig. 2. Physical design of the PLL with area 525×326 μm.

SPICE simulation results of the circuit are shown in Fig. 3, which shows the frequency over time plot, with the PLL locking at 24.58 μs. The baseline phase noise diagram in Fig. 4 shows that the circuit has acceptable phase noise, (-163 dBc/Hz at 10 Hz offset). The average power consumption of the PLL is 9.28 mW.

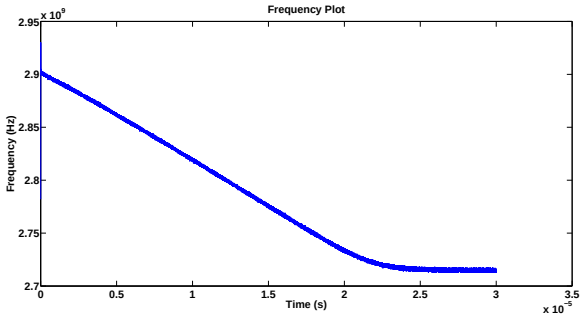

Fig. 3. Frequency plot for the PLL circuit.

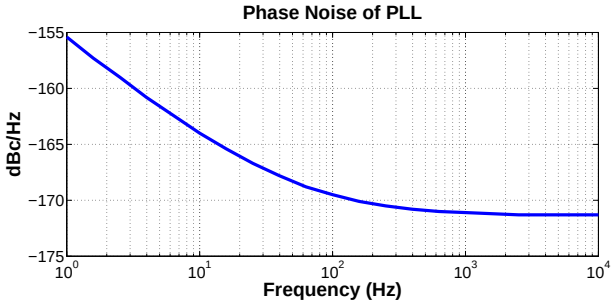

Fig. 4. Phase noise of the baseline PLL circuit.

IV. Non-Polynomial Metamodeling of the PLL

A. Metamodeling Design Flow

The proposed design flow using non-polynomial metamodels is shown in Fig. 5. The physical design is parameterized and used twice, once for training samples and once more for verification. The non-polynomial metamodels are created for each output set of the design. Computationally expensive optimization algorithms can be applied using the fast non-polynomial metamodels as they are ultra fast compared to the actual RCLK netlist. The optimized values are then used to adjust the initial physical layout to create the near optimal design. This design flow only uses 2 iterations for physical design, at the beginning and the end. Overall, the design flow is as accurate as the parasitic-aware netlist of the circuit but ultra-fast due to the metamodel abstraction, which in turn minimizes the amount of time the designer needs to spend on the design optimization of the circuit.

B. Neural Network Exploration for Non-Polynomial Metamodeling

Neural network models are composed of a mass of fairly simple computational elements and rich interconnections between the elements. Neural networks operate in a parallel and distributed fashion which may resemble biological neural networks. Most neural networks have some sort of "training" rule by which the weights of connections are adjusted on the basis of presented patterns. They normally have great potential for parallelism, since the computations of the components are independent of each other. It has been proven in the universal approximation theorem that a neural network with one hidden layer can estimate any continuous function that maps to real numbers.

Over-fitting is the phenomenon where the network becomes worse instead of improving after a certain point during training when it is trained to as low errors as possible. This is because excessive training or a large amount of neurons in the hidden layer may make the network memorize the training patterns and stop adjusting the weights. There are several methods to avoid over-fitting. One method is regularization which tries to limit the complexity of the network such that it is unable to learn peculiarities. Another method is early stopping which aims to stop training at the point of optimal generalization.

1) Multilayer Neural Networks: A multiple layer neural network consists of an input, a with nonlinear activation function in hidden layer, and linear activation function in the output layer. Multilayer networks are very flexible and powerful due to their ability to represent nonlinear as well as linear functions. The multilayer network needs to have at least one non-linear function, otherwise a composition of linear functions becomes just another linear function.

The two common nonlinear activation functions that are usually used for the hidden layer are [18]:

$$b_j\left(v_j\right) = \left(\frac{1}{1+e^{-\lambda v_j}}\right), \text{ or} \tag{1}$$

$$b_j\left(v_j\right) = \tanh(\lambda v_j), \tag{2}$$

where j denotes a neuron in the hidden layer, b_j and v_j are its input and output, respectively, and λ is the neuron transfer function steepness. The predicted output is given by:

$$\hat{y} = \sum_{j=1}^{d} \beta_j b_j(v_j) + \beta_0, \tag{3}$$

where β_j is the weight in the output due to neuron j, d is the number of neurons in the hidden layer and β_0 a constant. On the other hand, a linear layer function has the following format:

$$v_i = \sum_{i=1}^{s} w_{ji} x_i + w_{j0}, \tag{4}$$

Where w_{ji} is the weight connection between the jth component in the hidden layer and the ith component of the input.

The network training is performed to minimize the least square criterion:

$$E = \sum_{k=1}^{n} (y_k - \hat{y_k})^2, \tag{5}$$

where y_k and $\hat{y_k}$ are the actual and predicted responses, respectively, at the k-th training point (of n).

2) Radial Neural Networks: Radial neural networks are also two-layer networks. The first layer has radial base neurons, and calculates its weighted inputs with distance and its net input with a radial function. The second layer has linear neurons, and calculates its weighted input with a dot product function and its net inputs by combining its weighted

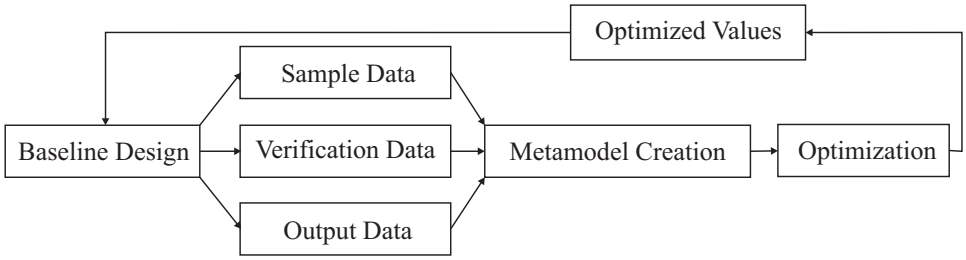

Fig. 5. The non-polynomial metamodeling based design flow.

inputs and biases. Both layers have biases. The radial network mathematical model is as follows:

$$y = \sum_{i=1}^{N} a_i \rho \left(\| x - c_i \| \right), \tag{6}$$

where c_i is the center vector of neuron i, x is the prediction point, ρ is the neuron's transfer function and a_i are the weighs of the linear neuron.

Initially the radial basis layer has no neurons. The following steps are repeated until the network's mean squared error falls below the desired goal. The network is simulated. The input vector with the greatest error is found. A radial basis neuron is added with weights equal to that vector. The linear layer weights are then recalculated to minimize the error.

Each neuron in the radial basis layer will output a value according to how close the input vector is to each neuron's weight vector. Thus, radial basis neurons with weight vectors quite different from the input vector have will have outputs near to zero. These small outputs have only a negligible effect on the linear output neurons.

In contrast, a radial basis neuron with a weight vector close to the input vector produces a value near 1. If a neuron has an output of 1 its output weights in the second layer pass their values to the linear neurons in the second layer.

C. Standardizing or Normalizing Data

The data set is generated from the RLCK parasitic aware netlist simulations. The input data set is the same for every metamodel and is generated using Latin Hyper Cube Sampling (LHS). LHS supports any amount of planes and is proven to work better than Monte Carlo due to the more even distribution of points with still the random factor that helps to detect nonlinearity. LHS divides each plane (parameter) into Latin squares and randomly picks a point from each square. Output is generated for each run from SPICE simulations saving each needed value to its own data set. Hence, each metamodel has its own target data set. This paper targets neural networks that have a single output with multiple inputs.

The validation and test data must be standardized or normalized using the statistics computed from the training data. It is desirable to either normalize or standardize the input data as the input dataset has large dynamic range. If not, the training of higher values can outweigh the lower and the neural network will not train properly. In this paper, 2 commonly used methods are applied to standardize the data:

1) Normalizing to mean (μ) 0 and standard deviation (σ) 1.
2) Standardizing to midrange 0 and range 2 (from -1 to 1).

D. Metamodel Generation from Trained Neural Network Data

For comparison purposes, the data was fitted into partial polynomial equations. Since the full polynomial function would result in a very large amount of coefficients for 21 variables, partial polynomial functions of order of 1 through 6 are considered. Further, the stepwise regression method is used to filter out the coefficients that do not contribute to the function's outcome.

E. Metamodel Selection Criteria

There may be numerous metamodels created from the same sampled set. RMSE and R^2 are common metrics used for goodness of fit. The Root Mean Square Error (RMSE) is derived from sum of square errors (SSE):

$$RMSE = \sqrt{\frac{1}{N} SSE} \tag{7}$$

$$= \sqrt{\frac{1}{N} \sum_{k=1}^{N} \left(y(x_k) - \hat{y}(x_k) \right)^2}. \tag{8}$$

Where y are the actual simulation result values and $\hat{y}$ are the results of the metamodel at the same location as the simulation point. R^2 is the coefficient of determination, which predicts the probability of a future result to be predicted by the model and is also used to verify the model accuracy.

The created model may fit perfectly to the training data set but may not qualify as a good model to represent the output for the given process at other points. For this reason, the verification data set is created so that the points are at different locations than the original sample. It is a good idea not to use the verification set for training, since it will defeat the purpose of testing the metamodel on totally unbiased points. If the verification RMSE and R^2 values do not differ very much from the training values, then the model has been trained correctly, otherwise it has not.

F. Design Optimization of the PLL Circuit

The best (may be the most accurate or the fastest, depending on the requirement) non-polynomial metamodels from the previous section need to be selected for optimization. The optimization algorithm that is being used is the Bee Colony

Algorithm (BCA). BCA is the artificial representation of a bee colony behavior as bees try to find the best food source [19], [20]. More information about the algorithm in the context of AMS circuit optimization can be found in previous research [17]. This algorithm was found to be effective for use on AMS circuits with metamodels. As a specific objective and constraint optimization, the PLL circuit is characterized for output frequency, power, and locking time. A separate metamodel is created for each Figure-of-Merit (FoM) from the same sample set. Each single simulation calculates all FoMs so the number of simulations that are needed does not depend on the number of the metamodels that need to be created.

V. Experimental Results

Given that each SPICE simulation for the PLL circuit takes approximately 10 minutes to converge, the amount of simulation runs are limited. In this work 100 simulations for training and 30 simulations for verification have been chosen. Different architectures of neural networks are evaluated. For feed-forward networks two differentiable transfer functions (tanh - tansig, and logarithmic - logsig) are used for the hidden layer. In addition, the experimental results also consider the difference between raw regular input data in comparison to normalized and standardized input sets.

The verification data set is also chosen using LHS, but it is ensured that none of the points match the training set. After the neural network training is completed, the input values for verification set are fed into the network and the RMSE value is calculated for the verification set. The R^2 values are calculated for training and verification sets for each combination of the above neural networks. Selected results are summarized in Tables I and II for brevity. The statistics of the best created polynomial functions that were created from [1] are listed in the last rows of the tables for comparison purposes.

From the data it is observed that neural networks with no standardization of the input data perform the worst. Even though polynomials show best results without standardization or normalization, this is not the case for neural networks. Also, it can be concluded that neural networks perform better fitting for this circuit, mostly because of the non-linear and linear flexibility of the neural networks. The data also demonstrates which architecture and normalization or standardization should be used, i.e. which has the best performance.

Fig. 6 shows the progression of the FoM as the BCA optimization progresses. The frequency metamodel is used to filter (constraint) the results within 0.05% of the required 2.7 GHz operational locking frequency of the PLL, which can be used in Multichannel Multipoint Distribution Systems (MMDS). The constraint narrows down the search criteria for the algorithm. If the model is within range for frequency, the power and locking time metamodels are used to calculate a composite FoM, which is defined by:

$$FoM = \left(\frac{1}{power \times lockingTime} \right). \quad (9)$$

Table III shows the optimized values for both polynomial and neural network metamodels.

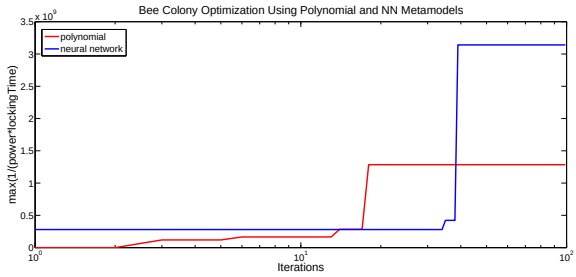

Fig. 6. Bee Colony optimization conducted on polynomial and neural network metamodels for optimizing power and locking time within 0.5% of 2.7 GHz frequency constraint.

TABLE III
PLL Circuit Parameters after Optimization.

FoM	Polynomial	Neural Network
Power	3.9 mW	3.9 mW
Locking Time	8.476 μs	3.3147 μs
Frequency	2.6909 GHz	2.7026 GHz

VI. Conclusion and Future Research

This paper explored the generation and usage of neural networks for metamodels of a PLL circuit. The bee colony algorithm with both non-polynomial and polynomial metamodels has been used for optimization. Neural networks are reusable and can be used as a system of equations to accurately represent the needed output. Neural networks show on average 56% increase in accuracy of prediction over the polynomial metamodels that have been generated from the same input data samples. In addition, neural network prediction, which is on average within 3.2% of SPICE output, is enormously faster than SPICE simulation and is shown to find better solution during the optimization phase of design. Even though the circuit that this paper uses as an example is parameterized with 21 parameters, in future work higher and more complex circuits that can have hundreds of parameters will be investigated.

References

[1] O. Garitselov, S. P. Mohanty, and E. Kougianos, "Nano-CMOS Mixed-Signal Circuit Metamodeling Techniques: A Comparative Study," in *Proc. Int. Symp. Elec. Des.*, 2010, pp. 191–196.

[2] S. Basu, B. Kommineni, and R. Vemuri, "Variation-Aware Macromodeling and Synthesis of Analog Circuits Using Spline Center and Range Method and Dynamically Reduced Design Space," in *Proc. Int. Conf. VLSI Des., VLSID*, 2009, pp. 433–438.

[3] R. Barton, "Simulation Optimization Using Metamodels," in *Proceedings of the 2009 Winter Simulation Conference (WSC).*, December 2009, pp. 230–238.

[4] L. Wang, "A Hybrid Genetic Algorithm- Neural Network Strategy for Simulation Optimization," *Applied Mathematics and Computation*, vol. 170, no. 2, pp. 1329–1343, 2005.

[5] A. Doboli and R. Vemuri, "Exploration-based High-level Synthesis of Linear Analog Systems Operating at low/medium Frequencies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 11, pp. 1556–1568, November 2003.

[6] S. Roy, W. Chen, C. Chung-Ping Chen, and Y. H. Hu, "Numerically Convex Forms and Their Application in Gate Sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1637–1647, September 2007.

TABLE I
FREQUENCY NON-POLYNOMIAL METAMODEL COMPARISON OF THE PLL.

Function	Data Filtering	R^2-test	R^2- verification	RMSE	Neurons
logsig→purelin	none	0.802	0.723	52.74 MHz	4
tansig→purelin	none	0.839	0.713	51.24 MHz	3
radial→purelin	none	0.020	0.490	81.51 MHz	
logsig→purelin	minmax	0.917	0.664	48.89 MHz	9
tansig→purelin	minmax	0.855	0.699	53.65 MHz	1
radial→purelin	minmax	0.844	0.712	50.88 MHz	
logsig→purelin	meanstd	0.843	0.733	53.60 MHz	1
tansig→purelin	meanstd	0.793	0.762	51.64 MHz	5
radial→purelin	meanstd	0.848	0.749	48.97 MHz	
	Data Filtering	R^2	Order	RMSE	Number of Coefficients
polynomial	none	0.930	4	77.96 MHz	48

TABLE II
LOCKING TIME NON-POLYNOMIAL METAMODEL COMPARISON OF THE PLL.

Function	Data Filtering	R^2-Test	R^2-Verification	RMSE	Neurons
logsig→purelin	none	0.828	0.873	1.30 μs	1
tansig→purelin	none	0.850	0.723	1.44 μs	9
radial→purelin	none	0.078	0.830	2.26 μs	
logsig→purelin	minmax	0.826	0.870	1.29 μs	1
tansig→purelin	minmax	0.839	0.942	1.12 μs	10
radial→purelin	minmax	0.931	0.508	1.65 μs	
logsig→purelin	meanstd	0.826	0.906	1.22 μs	2
tansig→purelin	meanstd	0.737	0.939	1.12 μs	3
radial→purelin	meanstd	0.963	0.691	1.23 μs	
	Data Filtering	R^2	Order	RMSE	Number of Coefficients
polynomial	none	0.877	4	1.91 μs	56

[7] A. Pradhan and R. Vemuri, "A Layout-aware Analog Synthesis Procedure Inclusive of Dynamic Module Geometry Selection," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, ser. GLSVLSI '08. New York, NY, USA: ACM, 2008, pp. 159–162.

[8] D. Ghai, S. P. Mohanty, and E. Kougianos, "Design of Parasitic and Process-Variation Aware Nano-CMOS RF Circuits: A VCO Case Study," *IEEE Trans. VLSI Syst.*, vol. 17, no. 9, pp. 1339–1342, 2009.

[9] G. Wolfe and R. Vemuri, "Extraction and Use of Neural Network Models in Automated Synthesis of Operational Amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, vol. 22, no. 2, pp. 198–212, February 2003.

[10] L. Xia, I. Bell, and A. Wilkinson, "A Robust Approach for Automated Model Generation," in *DTIS '09. 4th International Conference on Design Technology of Integrated Systems in Nanoscal Era, 2009.*, April 2009, pp. 281–286.

[11] P. Zhongliang, "Neural Network Model for Testing Stuck-at and Delay Faults in Digital Circuit," in *Proceedings 17th International Conference on VLSI Design, 2004.*, 2004, pp. 499–504.

[12] Y. Cao and Q.-J. Zhang, "Neural Network Techniques for Fast Parametric Modeling of Vias on Multilayered Circuit Packages," in *2010 IEEE Electrical Design of Advanced Packaging Systems Symposium (EDAPS)*, December 2010, pp. 1–4.

[13] A. Zaabab, Q.-J. Zhang, and M. Nakhla, "A Neural Network Modeling Approach to Circuit Optimization and Statistical Design," *IEEE Transactions on Microwave Theory and Techniques.*, vol. 43, no. 6, pp. 1349–1358, June 1995.

[14] E. Macii and M. Poncino, "Estimating Power Consumption of CMOS Circuits Modelled as Symbolic Neural Networks," *IEE Proceedings Computers and Digital Techniques.*, vol. 143, no. 5, pp. 331–336, September 1996.

[15] J. Xu, M. Yagoub, R. Ding, and Q. Zhang, "Feedforward Dynamic Neural Network Technique for Modeling and Design of Nonlinear Telecommunication Circuits and Systems," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 2, July 2003, pp. 930–935.

[16] Z. Aimin, Z. Hang, L. Hong, and C. Degui, "A Recurrent Neural Networks Based Modeling Approach for Internal Circuits of Electronic Devices," in *2009 20th International Zurich Symposium on Electromagnetic Compatibility*, January 2009, pp. 293–296.

[17] O. Garitselov, S. P. Mohanty, E. Kougianos, and P. Patra, "Bee Colony Inspired Metamodeling Based Fast Optimization of a Nano-CMOS PLL," in *Proceedings of the 2nd IEEE International Symposium on Electronic System Design (ISED)*, 2011.

[18] K.-T. Fan, R. Li, and A. Sudjianto, *Design and Modeling for Computer Experiments*. 23-25 Blades Court, London SW15 2NU, UK: Chapman and Hall/CRC, 2006.

[19] D. Karabora and B. Akay, "A Comparative Study of Artificial Bee Colony Algorithm," *Applied Mathematics and Computation*, no. 214, pp. 108–132, 2009.

[20] R. Hedayatzadeh, B. Hasanizadeh, R. Akbari, and K. Ziarati, "A Multi-Objective Artificial Bee Colony for Optimizing Multi-Objective Problems," in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, vol. 5, August 2010, pp. 277–281.