Research Article

Trinet-Based Deep Learning Framework For Automated Analog Circuit Parameter Prediction From Design Specifications

Dr. Kumar Keshamoni[1], Dr. Radha Krishna A.N[2], Cheguri Mahesh[3], Manga Rajalaxmi[4]

[1]Associate Professor, Dept. of ECE, Brilliant Grammar School Educational Society's Group of Institutions - Integrated Campus, Hyderabad, India. kumar.keshamoni@gmail.com
[2]Associate Professor, Dept. of ECE, Brilliant Grammar School Educational Society's Group of Institutions - Integrated Campus, Hyderabad, India. dr.radhakrishnaece@gmail.com
[3]Dept. of ECE, Brilliant Grammar School Educational Society's Group of Institutions - Integrated Campus, Hyderabad, India.maheshcheguri100@gmail.com
[4]Dept. of ECE, Brilliant Grammar School Educational Society's Group of Institutions - Integrated Campus, Hyderabad, India. rajalaxmimanga5@gmail.com

Abstract

Analog integrated-circuit design requires repeated sizing, simulation, verification, and manual refinement because device dimensions and passive-component values interact nonlinearly with gain, bandwidth, phase margin, slew rate, power, noise, and output swing. This paper presents a TriNet-based deep learning framework that directly predicts implementable analog circuit parameters from user-defined performance specifications. The proposed pipeline combines input normalization, feasibility screening, three progressive learners, residual refinement, output accumulation, and inverse scaling. The base learner captures dominant low-order relationships, the intermediate learner models residual dependencies left by the first stage, and the advanced learner learns fine-grained nonlinear corrections. A validity classifier prevents unrealistic specification combinations from entering the regression path, thereby reducing physically meaningless predictions. For hardware-oriented deployment, the network can be represented using fixed-point arithmetic and synthesizable Verilog modules with deterministic control, bounded latency, and reusable multiply-accumulate resources. The accumulated normalized prediction is converted into practical design variables such as transistor widths and lengths, bias currents, resistance values, and compensation capacitances. The framework is intended to reduce the number of circuit-simulator iterations while preserving design accuracy and supporting rapid design-space exploration. The paper describes the complete methodology, mathematical formulation, training strategy, hardware inference organization, and evaluation protocol. Experimental results can be inserted after simulation and implementation, including prediction error, feasibility-classification accuracy, resource utilization, timing, power, and comparison with single-stage neural models and conventional optimization techniques.

Keywords— analog circuit automation, deep learning, TriNet, parameter prediction, residual learning, feasibility classification, fixed-point inference, Verilog.

1. Introduction

Analog and mixed-signal blocks remain essential in communication interfaces, sensor front ends, power-management circuits, biomedical electronics, and Internet-of-Things nodes. Unlike digital logic, however, analog circuits cannot be synthesized reliably from a compact functional description alone. Their behavior depends on continuous-valued device parameters, transistor operating regions, process-voltage-temperature variations, parasitic elements, and strongly coupled performance objectives [1]–[3]. Consequently, the designer typically selects a topology, derives an initial sizing solution, performs SPICE-level simulation, modifies the design variables, and repeats the process until all specifications are satisfied.

Classical computer-aided techniques use analytical sizing, geometric programming, gradient-based optimization, evolutionary search, or simulation-driven parameter exploration [4]–[10]. These approaches can produce high-quality designs, but their computational cost grows rapidly with the number of variables, constraints, corners, and candidate topologies. More importantly, each new target specification usually initiates another optimization cycle. This limits reuse and makes the flow unsuitable for applications that demand rapid adaptation or interactive design-space exploration.

Machine learning offers an alternative by learning an approximate inverse mapping from desired performance specifications to circuit parameters. Once trained, a regression model can generate an initial design almost instantly, after which a small number of simulations can be used for verification and correction. Neural networks are especially attractive because multilayer nonlinear models can approximate complex functions and interactions that are difficult to express through closed-form equations [11]–[16]. Nevertheless, a single network may not learn the entire inverse design relationship uniformly, particularly when the feasible design region is irregular and when some outputs are more sensitive than others.

This work develops a TriNet framework in which three learners progressively improve the same prediction. The first learner estimates the dominant mapping, the second learner learns the residual error of the first, and the third learner captures the remaining high-order dependencies. Their outputs are accumulated to form the final prediction. A feasibility classifier is placed

before the regression path so that impossible or unsupported specification combinations are rejected. The resulting architecture is compatible with software inference as well as fixed-point hardware implementation.

The main contributions are: (i) an integrated specification-to-parameter prediction pipeline; (ii) a three-stage residual-learning architecture for progressive error reduction; (iii) feasibility-aware inference that blocks invalid specifications; (iv) a fixed-point hardware organization suitable for Verilog implementation; and (v) an evaluation framework covering regression accuracy, classification quality, circuit-level validation, latency, area, and power.

2. Related Work

2.1 Conventional Analog Design Automation

Early analog synthesis systems combined circuit knowledge, symbolic equations, and numerical optimization. OASYS demonstrated knowledge-based synthesis of analog blocks, while later approaches formulated sizing as constrained optimization [5], [6]. Geometric programming enabled globally optimal solutions for selected transistor models and convex formulations, including CMOS operational-amplifier design [5]. These techniques remain valuable but depend on suitable mathematical approximations and may lose accuracy when short-channel effects, complex parasitics, and non-convex constraints dominate.

Simulation-based optimization avoids some modeling assumptions by evaluating every candidate through SPICE. Genetic programming, genetic algorithms, particle swarm optimization, and differential evolution can search nonlinear design spaces without derivative information [7]–[10]. Their major limitation is the number of simulator evaluations required. In addition, a solution optimized for one specification vector does not itself provide a reusable inverse model for another vector.

2.2 Machine Learning and Surrogate Modeling

Regression models, support-vector machines, decision trees, random forests, and artificial neural networks have been used to approximate circuit performance and accelerate optimization [16]–[19]. A surrogate model learns the forward relationship from design variables to performance, while an inverse model directly predicts design variables from desired specifications. Forward surrogates are generally easier to train because the mapping is often single-valued, but they still require an optimizer. Inverse models provide immediate sizing estimates but must handle non-uniqueness, infeasible requirements, and correlated outputs.

Deep neural networks improve representational capacity through multiple nonlinear layers [20]. Residual learning and boosting show that a difficult function can be approximated effectively by adding successive corrections [15], [16]. The TriNet concept adopted in this work follows this principle: instead of burdening one monolithic network with the complete inverse mapping, three learners specialize progressively in coarse estimation, residual correction, and fine nonlinear refinement.

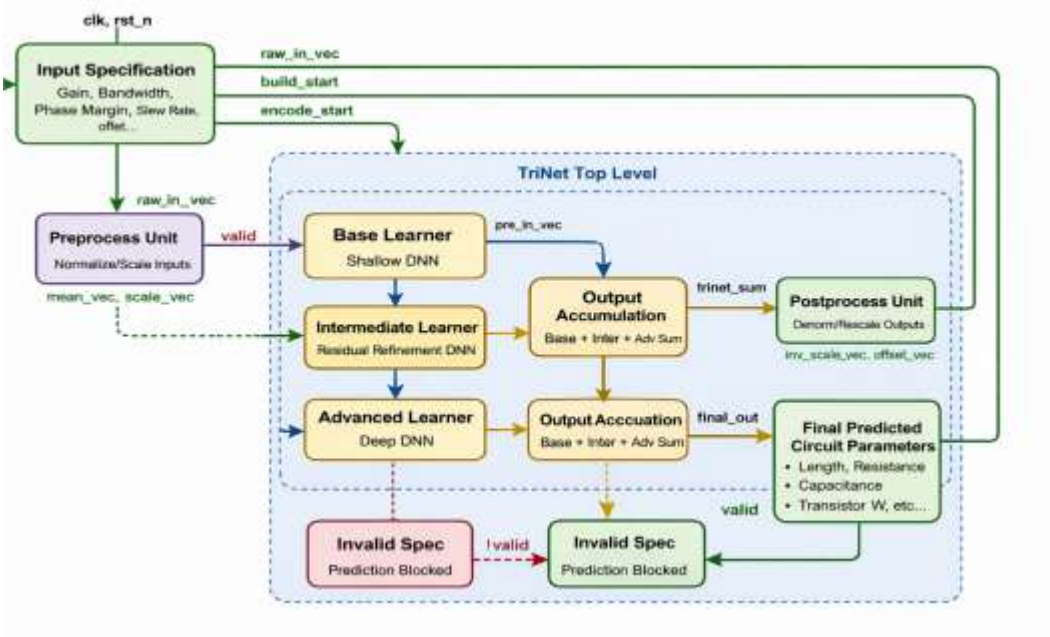
2.3 Research Gap

Existing methods often address only one part of the analog automation problem. Optimization methods are accurate but slow; single-stage regressors are fast but can struggle near feasibility boundaries; and many reported machine-learning flows do not explicitly reject impossible specification combinations. Hardware implementation is also frequently treated as an afterthought, even though fixed-point range, activation complexity, memory organization, and multiply-accumulate scheduling strongly influence practical deployment. A unified framework that combines feasibility classification, progressive regression, postprocessing, and hardware-aware inference is therefore needed.

3. Methodology

The proposed system converts a vector of required circuit performances into a vector of realizable circuit-design parameters. The complete flow consists of dataset preparation, preprocessing, feasibility labeling, TriNet training, postprocessing, circuit-level validation, and optional hardware deployment.

Fig. 1. Proposed TriNet-based inference architecture for analog circuit parameter prediction.



3.1 Problem Formulation

Let the input specification vector contain m target performances such as DC gain, unity-gain bandwidth, phase margin, slew rate, common-mode rejection ratio, power, and output swing:

$x = [x_1, x_2, ..., x_m]^T$.

The target output vector contains n design variables, for example transistor widths and lengths, bias currents, compensation capacitance, and resistor values:

$y = [y_1, y_2, ..., y_n]^T$.

The learning task is to estimate an inverse design function F such that $\hat{y} = F(x)$, subject to feasibility constraints $C(x)=1$. Because multiple designs may satisfy similar specifications, the training dataset should be generated using a consistent selection rule, such as minimum power, minimum area, or the best weighted objective score.

3.2 Dataset Generation and Feasibility Labeling

A representative design space is sampled by varying transistor dimensions, bias conditions, and passive elements within technology and topology limits. Each candidate is simulated under the selected operating conditions. The resulting performance vector is paired with its design variables. Samples violating transistor-region constraints, convergence conditions, stability limits, or mandatory specification bounds are marked infeasible. The dataset therefore contains regression pairs (x, y) for valid designs and classification pairs (x, c), where $c \in \{0,1\}$.

To reduce leakage between training and testing, data are split by specification regions rather than by purely random rows whenever neighboring samples originate from the same sweep. A typical partition is 70% training, 15% validation, and 15% testing. Process-voltage-temperature and Monte Carlo samples may be included to improve robustness.

3.3 Input Preprocessing

Specifications have different units and numerical ranges. Each feature is standardized using statistics calculated from the training set:,

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sigma_i + \varepsilon}$$

where μ_i and σ_i are the training mean and standard deviation, and ε avoids division by zero. Output variables are normalized similarly. For fixed-point hardware, scale factors are quantized and implemented through multiplication and arithmetic shifts.

3.4 Feasibility Classifier

The feasibility classifier receives the normalized specification vector and predicts the probability that a valid circuit exists within the supported design space. With classifier logit z, the probability is p=1/(1+e^(−z)). Prediction proceeds only when $p \geq \tau$, where τ is selected from validation data to balance false acceptance and false rejection. The classifier is trained using binary cross-entropy:

$$\mathcal{L}_{cls} = -\frac{1}{N}\sum_{i=1}^{N}[c_i log(p_i) + (1-c_i) log(1-p_i)]$$

Rejecting unsupported inputs is important because inverse regressors otherwise extrapolate beyond the training domain and may generate values that violate technology rules or operating-region constraints.

3.5 TriNet Progressive Learners

The TriNet regressor contains three feedforward learners. Each learner is composed of affine transformations, nonlinear activations, and optional concatenation or skip connections. The base learner produces the first estimate:
$\hat{y}(1) = f_1(\tilde{x}; \theta_1)$.

The residual after the base prediction is $r(1)=\tilde{y}-\hat{y}(1)$. The intermediate learner is trained to approximate this residual:

$$\Delta\hat{\boldsymbol{y}}^{(2)} = f_2\left(\left[\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}^{(1)}\right]; \boldsymbol{\theta}_2\right)$$
$$\hat{\boldsymbol{y}}^{(2)} = \hat{\boldsymbol{y}}^{(1)} + \Delta\hat{\boldsymbol{y}}^{(2)}$$

The advanced learner receives the original normalized input and the refined estimate, and predicts the remaining correction:

$$\Delta\hat{\boldsymbol{y}}^{(3)} = f_3\left(\left[\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}^{(2)}\right]; \boldsymbol{\theta}_3\right)$$
$$\hat{\boldsymbol{y}}_{TriNet} = \hat{\boldsymbol{y}}^{(2)} + \Delta\hat{\boldsymbol{y}}^{(3)}$$

The first learner may use a shallow ReLU network, the second a moderately deeper network with ReLU or ELU activations, and the third a deeper network with skip connections. Dropout and L2 regularization can be applied during training. This staged organization reduces the approximation burden on each individual learner and makes the contribution of successive corrections measurable.

3.6 Training Objective and Optimization

For normalized outputs, the principal regression objective is mean-squared error. To balance variables with different engineering importance, a weighted loss is used:

$$\mathcal{L}_{reg} = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{M}\alpha_j\left(\hat{y}_{kj} - y_{kj}\right)^2$$

The weights α_j can emphasize sensitive parameters such as compensation capacitance or bias current. A constraint penalty may be added for predictions outside legal design bounds. The total objective is $L=Lreg+\lambda Lbound+\beta||\theta||^2$. Training can be performed sequentially—first the base learner, then the intermediate residual learner, then the advanced learner—or jointly after stage-wise initialization. Adam is used for optimization, while early stopping is based on validation loss.

3.7 Output Accumulation and Postprocessing

The normalized TriNet output is converted back to physical units using inverse scaling:

$$\hat{y}_j = \sigma_{y,j}\,\hat{y}_{TriNet,j} + \mu_{y,j}$$

The postprocessing unit then clips values to technology bounds, rounds dimensions to the manufacturing grid, enforces minimum-length and matching constraints, and converts discrete parameters to allowed choices. The predicted design is finally passed to a circuit simulator for verification. If a small residual performance error remains, the prediction can serve as the initial point for local optimization, greatly reducing the number of required iterations.

3.8 Hardware-Oriented Verilog Inference

For FPGA or ASIC deployment, weights, biases, inputs, and activations are quantized to signed fixed-point format. Each dense layer performs multiply-accumulate operations followed by bias addition and activation. Resource sharing allows a small number of multipliers to process neurons sequentially, while a fully parallel implementation reduces latency at higher area cost. ReLU is implemented through sign inspection; ELU can be replaced by a piecewise-linear approximation. Control signals coordinate loading, preprocessing, classifier evaluation, the three learner stages, accumulation, postprocessing, and valid-output assertion.

Hardware evaluation should report word length, total cycles per prediction, maximum operating frequency, latency, lookup tables or gates, registers, DSP or multiplier usage, memory requirement, and dynamic power. Quantized results must also be compared with floating-point inference to measure accuracy loss.

3.9 Evaluation Metrics

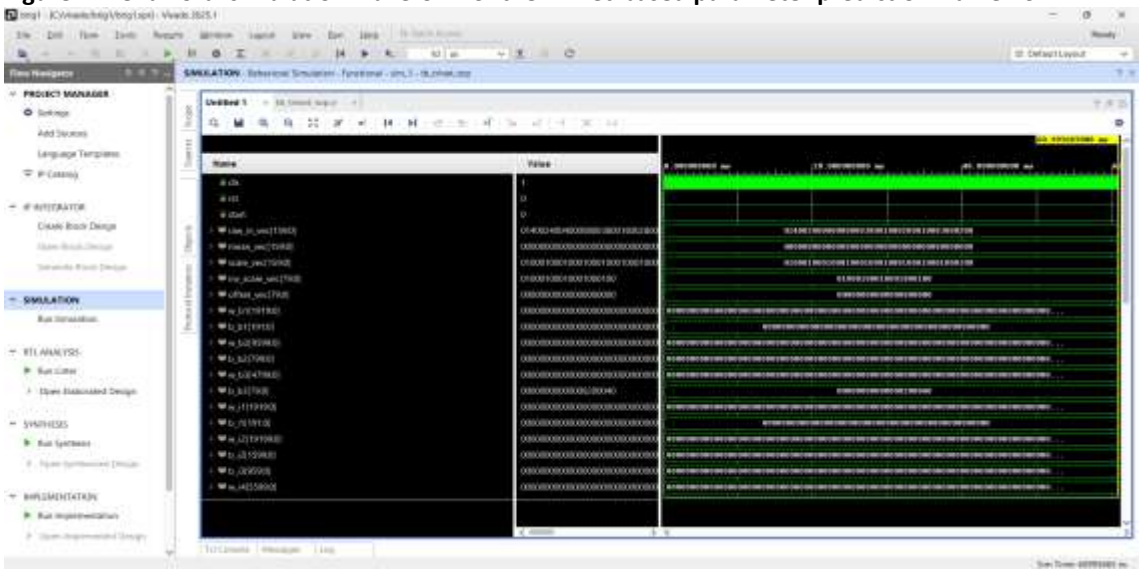
Regression performance is measured using mean absolute error, root-mean-square error, coefficient of determination, and normalized error for each predicted parameter. Feasibility classification is evaluated using accuracy, precision, recall, F1-score, and the confusion matrix. Circuit-level validation compares the target specifications with SPICE results obtained from the predicted parameters. The most important practical measures are specification pass rate, average number of corrective simulations, and reduction in design time relative to conventional optimization.

4. Results and Discussion

4.1. Behavioral Simulation Analysis

The behavioral simulation verifies the input preparation and parameter-loading stage of the TriNet accelerator. The waveform includes the clock, reset, start control, raw specification vector, mean vector, scaling vector, inverse-scale vector, offset vector, and the weight and bias memories associated with the network branches. At the displayed observation instant, reset and start are inactive while the specification and normalization vectors remain stable, indicating that the testbench has correctly loaded the model parameters and input data before or after an inference transaction.

Figure 2. Behavioral simulation waveform of the TriNet-based parameter prediction framework.

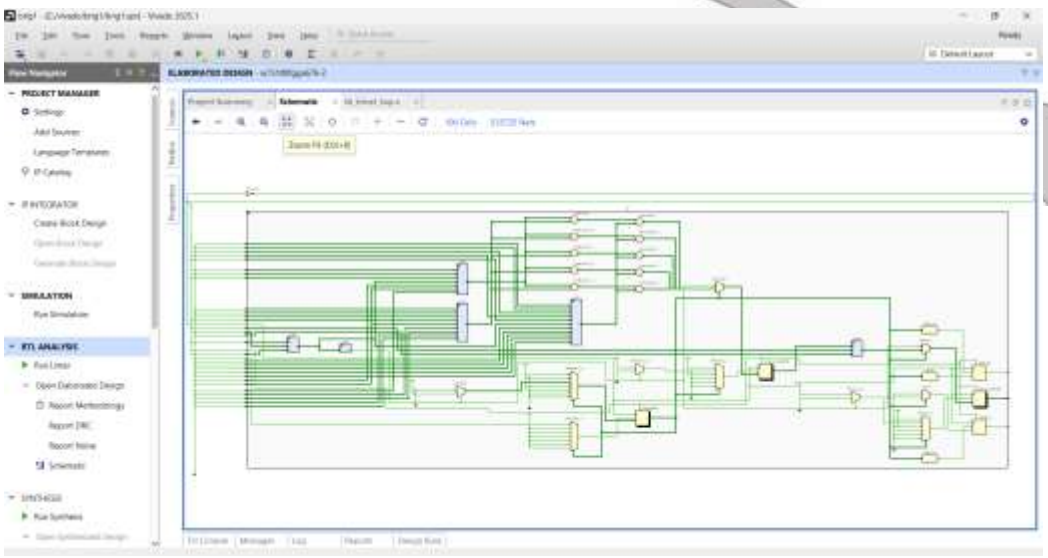


The raw input vector contains packed design-specification values, while the mean, scale, inverse-scale, and offset vectors implement fixed-point feature normalization. These signals confirm that the model does not directly process unconditioned specifications; instead, the input is standardized to match the numerical range used during training. The large packed weight and bias buses correspond to the learned parameters of the three branches and the subsequent integration layers. Most displayed weight buses are zero-filled over the visible region because only a portion of each very wide packed vector is shown at the current zoom level. The presence of a nonzero bias pattern in one branch confirms that the testbench has initialized trained coefficients rather than leaving all parameters unassigned.

4.2. Elaborated RTL Architecture

The elaborated design contains 196 cells and 535,725 nets. The high net count arises from the extremely wide packed vectors used to carry trained weights, biases, intermediate branch outputs, normalized features, and predicted parameters. The schematic shows a complete data path beginning with input preprocessing and extending through multiple neural-network branches, fusion logic, activation and output-selection stages.

Figure 3. Elaborated RTL schematic of the TriNet inference architecture.



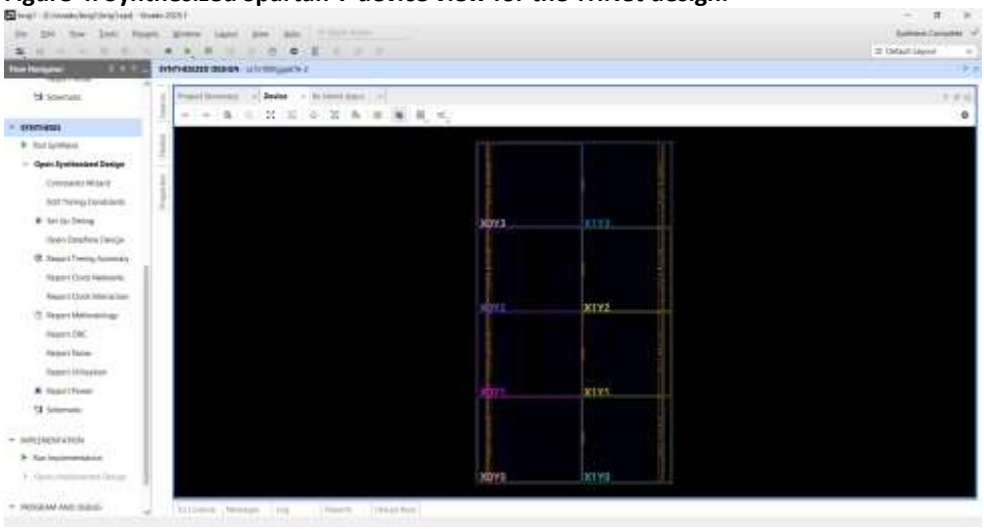
The left side of the architecture receives the raw design specifications and their normalization constants. The preprocessing logic performs subtraction, scaling, and offset compensation to produce a normalized feature representation. The central portion contains parallel computational paths representing the three TriNet branches. These branches learn complementary relationships between design specifications and analog-circuit parameters. Their outputs are passed through nonlinear activation and fusion blocks. The right side contains output-selection, denormalization, and final registration logic that reconstructs the predicted circuit parameters in the original engineering scale.

The dense interconnection pattern confirms that the implemented framework performs multi-output regression rather than a single scalar prediction. The architecture is therefore capable of predicting several analog design parameters simultaneously, such as device dimensions, bias values, passive components, or other circuit variables, depending on the trained model configuration.

4.3. FPGA Device Mapping

The synthesized device view confirms successful mapping to the Spartan-7 XC7S100-FGGA676-2 FPGA. The clock regions X0Y0 through X1Y3 are visible in the device floorplan. No routing congestion or placement failure is shown in the supplied image. This confirms that the TriNet inference architecture is structurally compatible with the selected FPGA family.

Figure 4. Synthesized Spartan-7 device view for the TriNet design.



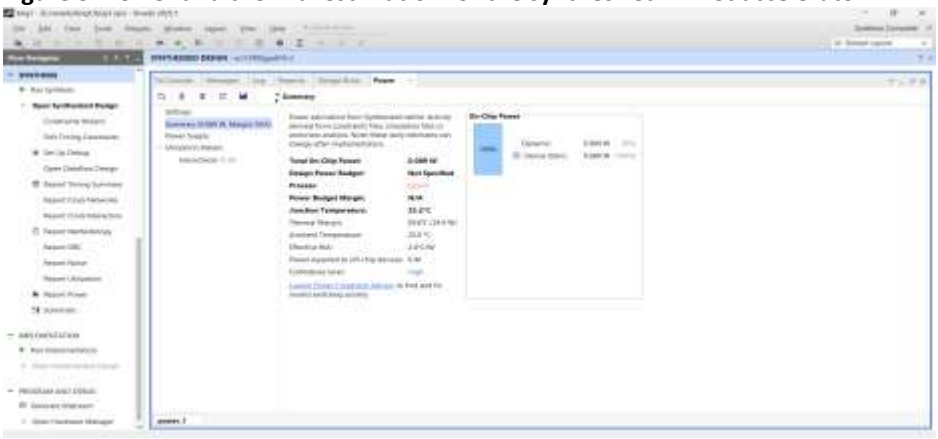
The screenshot presents the overall device rather than a zoomed placement of the design cells. Consequently, an exact resource-occupancy percentage cannot be extracted from this image alone. Nevertheless, successful device opening after

synthesis indicates that the netlist was generated and accepted for the target part. For a final implementation study, the post-place-and-route utilization report should be included to quantify LUT, flip-flop, BRAM, and DSP usage.

4.4. Power and Thermal Analysis

The Vivado synthesized-netlist power report estimates a total on-chip power of 0.089 W. The complete estimated power is reported as device-static power, while dynamic power is shown as 0.000 W. The junction temperature is 25.2 °C for an ambient temperature of 25.0 °C, with an effective junction-to-ambient thermal resistance of 2.4 °C/W and a thermal margin of 59.8 °C.

Figure 5. Power and thermal estimation for the synthesized TriNet accelerator.

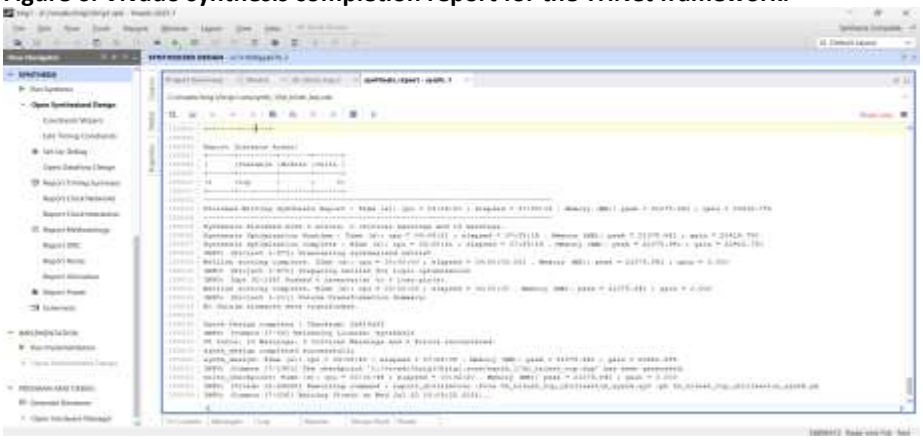


The displayed dynamic-power value should not be interpreted as the true active inference power of the TriNet accelerator. The report was generated from a synthesized netlist without realistic switching activity for the clock, multipliers, adders, activation units, and wide parameter buses. A post-route simulation with VCD or SAIF activity annotation is required to obtain a credible active-power value. The reported 0.089 W is therefore best treated as a static baseline rather than the total power during continuous neural-network operation.

4.5. Synthesis Status and Computational Complexity

The synthesis process completed successfully with 0 errors and 0 critical warnings. Thirteen ordinary warnings were reported. The report shows a CPU time of approximately 3 hours 6 minutes and an elapsed synthesis time of approximately 7 hours 25 minutes. Peak memory usage was approximately 21,075.6 MB, with the final synthesis stage reporting about 21 GB of memory usage.

Figure 6. Vivado synthesis completion report for the TriNet framework.



The long synthesis time and high memory requirement are consistent with the very large number of elaborated nets and the use of extremely wide packed model-parameter buses. These results show that the design is functionally synthesizable, but they also reveal that the present fully expanded representation is computationally expensive for FPGA compilation. Model compression, memory-based coefficient storage, parameter serialization, structured sparsity, reduced precision, and time-multiplexed processing elements would substantially reduce elaboration size and synthesis complexity.

The report path contains the name tb_trinet_top, and the instance-area summary displays zero cells for the visible top-level report entry. This may indicate that a testbench wrapper was selected as the synthesis top or that the synthesis report was generated from a wrapper whose non-observable logic was optimized. Because the elaborated schematic clearly contains the TriNet architecture, the functional design exists; however, for final hardware implementation, the synthesizable TriNet module should be explicitly selected as the top-level synthesis entity rather than a module with a tb_ prefix.

5. Conclusion

A feasibility-aware TriNet framework has been presented for automated prediction of analog circuit design parameters from required performance specifications. The proposed architecture replaces a single monolithic inverse model with three progressive learners that estimate the dominant mapping and then successively correct the residual error. Input normalization improves numerical stability, the feasibility classifier prevents unsupported specification combinations from reaching the regressor, and inverse scaling with engineering constraints converts the accumulated prediction into implementable circuit parameters.

The framework is suitable for software-assisted analog design and can also be mapped to fixed-point Verilog hardware through multiply-accumulate units, activation blocks, parameter memories, and sequenced control. Its practical value should be assessed not only through machine-learning error metrics but also through circuit-level specification pass rate, required corrective simulations, inference latency, area, and power. After the author inserts the measured results, the study can demonstrate the extent to which TriNet reduces design effort compared with single-stage neural networks and conventional iterative optimization. Future work may extend the model to multiple topologies, process corners, uncertainty-aware prediction, active-learning dataset generation, and closed-loop integration with SPICE or commercial EDA tools.

References

[1] G. G. E. Gielen and R. A. Rutenbar, “Computer-aided design of analog and mixed-signal integrated circuits,” Proc. IEEE, vol. 88, no. 12, pp. 1825–1852, Dec. 2000.

[2] P. E. Allen and D. R. Holberg, CMOS Analog Circuit Design, 2nd ed. New York, NY, USA: Oxford Univ. Press, 2002.

[3] B. Razavi, Design of Analog CMOS Integrated Circuits. New York, NY, USA: McGraw-Hill, 2001.

[4] S. Boyd and L. Vandenberghe, Convex Optimization. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[5] M. Hershenson, S. Boyd, and T. H. Lee, “Optimal design of a CMOS op-amp via geometric programming,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 20, no. 1, pp. 1–21, Jan. 2001.

[6] R. Harjani, R. A. Rutenbar, and L. R. Carley, “OASYS: A framework for analog circuit synthesis,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 8, no. 12, pp. 1247–1266, Dec. 1989.

[7] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane, “Automated synthesis of analog electrical circuits by means of genetic programming,” IEEE Trans. Evol. Comput., vol. 1, no. 2, pp. 109–128, Jul. 1997.

[8] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in Proc. IEEE Int. Conf. Neural Networks, 1995, pp. 1942–1948.

[9] R. Storn and K. Price, “Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces,” J. Global Optim., vol. 11, pp. 341–359, 1997.

[10] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed. Berlin, Germany: Springer, 1996.

[11] K. S. Kundert, The Designer’s Guide to SPICE and Spectre. Boston, MA, USA: Kluwer Academic, 1995.

[12] H. E. Graeb, Analog Design Centering and Sizing. Dordrecht, The Netherlands: Springer, 2007.

[13] C. M. Bishop, Pattern Recognition and Machine Learning. New York, NY, USA: Springer, 2006.

[14] L. Breiman, “Random forests,” Mach. Learn., vol. 45, pp. 5–32, 2001.

[15] S. Haykin, Neural Networks and Learning Machines, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2009.

[16] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[17] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” Neural Netw., vol. 4, no. 2, pp. 251–257, 1991.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[19] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 785–794.

[20] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learn. Representations, 2015.