AI-assisted Design and Synthesis of Continuous-Time ADCs

Internal Report#1
AIR-CHIP Project (PID2022-138078OB-I00)

Author
Francisco Gómez Pulido

Distribution List
Gustavo Liñán-Cembrano
Jose M. de la Rosa

Date: February 19, 2025

Contents

1 TFG Pablo Díaz Lobo

Date: January 27, 2025

1.1 Introduction

Integrated circuits (ICs) are essential in electronic devices, and their complexity increases every year due to high demand. Although digital circuits dominate, analog circuits remain indispensable, especially in applications requiring high precision. However, the design of these circuits is still manual and lengthy, limiting their efficiency.

This work focuses on **Sigma-Delta analog-to-digital converters** ($\Sigma\Delta$), known for their versatility and efficiency in digitizing high-resolution, low-bandwidth signals, such as in audio and biomedical devices. A neural network-based approach is proposed to automate and optimize their design. These networks predict key variables directly from specifications, reducing iterations and accelerating the process. This inverse model leverages advanced artificial intelligence techniques, enhancing productivity and quality in IC design.

1.2 Analog-to-digital converters

Analog-to-Digital Converter (ADC): An electronic device that converts an analog (continuous) signal into a digital (discrete) signal.

Analog signals are continuous in time and amplitude, such as sound waves or temperature variations. However, computers and many digital systems can only process information in the form of numbers, i.e., digital data.

The conversion process is carried out in two main steps:

1. **Sampling**: The analog signal is taken at regular time intervals, known as the sampling rate.
2. **Quantization**: Each sample is rounded to the nearest digital value within a discrete range.

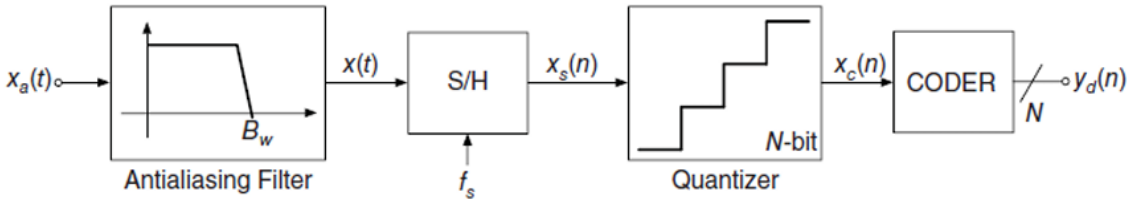

Figure 1: Conceptual block diagram

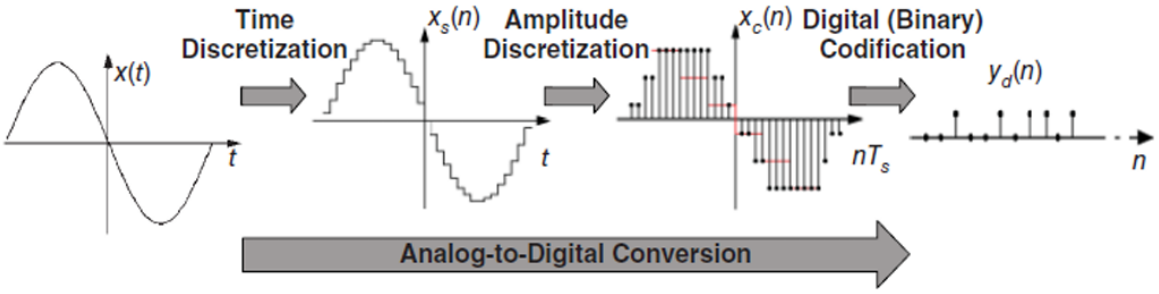

Figure 2: Conversion steps

Key components of ADCs:

1. **Anti-aliasing filter** (AAF): A low-pass filter that removes frequency components above half the sampling rate to prevent aliasing errors according to Nyquist's theorem.
2. **Sampler**: Converts the analog signal into a discrete-time signal.
3. **Quantizer**: Rounds the values of the sampled signal to discrete levels.
4. **Encoder**: Assigns a binary code to each quantized level, producing the final digital signal.

Important characteristics:

- **Sampling frequency** (f_s): Determines the number of samples taken per second. According to Nyquist's theorem, f_s must be at least twice the signal bandwidth (2Bw) to avoid information loss.
- **Resolution** (N): Indicates the number of bits used to represent each sample. The higher N, the greater the precision.
- **Signal-to-noise ratio** (SNR): Measures the quality of the conversion; a high SNR implies less noise in the resulting digital signal.

Sigma-Delta ($\Sigma\Delta$) ADCs are a particular type used when high precision and dynamic range are required, such as in digital audio, telecommunications, and precision sensors. The Sigma-Delta employs a special sampling and quantization technique that can be broken down into:

1. **Oversampling**: Instead of sampling at the minimum required frequency, the Sigma-Delta takes many more samples per second. This shifts noise to higher frequencies.
2. **Sigma-Delta Modulation**: This process converts the analog signal into a high-frequency digital sequence with only a few possible levels (e.g., 0 and 1).
3. **Digital filtering**: A digital filter then removes the noise introduced during modulation and reconstructs the digital signal with greater accuracy.

1.3 Neural networks

Neural networks (NNs) are computational models inspired by the functioning of the human brain. Their development began in 1943 with the work of McCulloch and Pitts, who introduced the concept of the neuron as a computational unit. However, initial interest declined after the publication of the book *Perceptrons* by Minsky and Papert in 1969, which highlighted the limitations of existing models. Decades later, thanks to theoretical advances such as the *backpropagation* algorithm and increased processing capacity, NNs resurged and became fundamental tools for solving complex problems in various fields, including integrated circuit design.

1.3.1 Fundamentals

NNs are composed of processing units, called neurons, connected through weights that determine the strength of the connections. Each neuron has an activation state, which is calculated based on the inputs it receives and an activation function. The main elements of a neural network include:

- **Neurons**: Units that process information.
- **Weights**: Coefficients that determine the influence of one neuron over another.
- **Activation state**: Represents the output of each neuron.
- **Propagation rule**: Computes the effective input of a neuron as the weighted sum of inputs plus a bias.
- **Activation functions**: Determine the neuron's output based on its effective input. Common functions include the sigmoid function:

$$y_k = \frac{1}{1 + e^{-s_k}} \tag{1}$$

  and the ReLU (Rectified Linear Unit):

$$y_k = \max(0, s_k) \tag{2}$$

- **Learning**: The process by which the network's weights are adjusted to minimize a loss function.

1.3.2 Connections between neurons

The connections between neurons determine how information is transmitted through the network. The effective input of a neuron is calculated as:

$$s_k(t) = \sum_j w_{jk} y_j(t) + \theta_k(t) \tag{3}$$

where w_{jk} is the connection weight, $y_j(t)$ is the output of neuron j, and $\theta_k(t)$ is the bias of neuron k.

1.3.3 Network topologies

There are different configurations or topologies for neural networks:

- **Feed-Forward Networks**: Connections move in a single direction, from input neurons to output neurons, passing through hidden layers. These are ideal for regression and classification problems.
- **Recurrent Networks**: Include feedback connections, allowing past outputs to influence future states. These are useful for tasks requiring memory, such as sequence processing.

In this work, *Feed-Forward* networks are used since they are better suited for regression problems where specifications are related to design variables.

1.3.4 Training

Training involves adjusting the network's weights to minimize the error between predicted and actual outputs, measured using a loss function. The basic steps are:

1. Forward propagation: Computes the network's outputs from the inputs.
2. Loss calculation: Evaluates the error using metrics such as mean squared error (MSE).
3. Backpropagation: Adjusts the weights by computing the loss gradient.
4. Weight update: Performed using algorithms such as Adam or SGD (*Stochastic Gradient Descent*).

The dataset is divided into two parts: training and validation. During training, data is processed in batches, and after completing a full pass through the dataset, an epoch is said to have elapsed.

1.3.5 Application to circuit design

In this work, neural networks are used to automate and optimize the design of Sigma-Delta modulators. Instead of the traditional approach, where multiple designs are tested until the desired specifications are met, an inverse approach is employed: the neural network directly predicts design variables from specifications. This accelerates the process and improves the quality of the final design.

1.4 Problem statement

Given a vector of specifications S_i (e.g., SNR, OSR, power consumption), the objective is to find the vector of design variables V_i that optimizes these specifications. For this purpose:

$$V = \arg\max P(V \mid S) \tag{4}$$

The model is trained using a dataset that relates valid designs to their specifications. Additionally, data augmentation, scaling, and normalization techniques are applied to improve network performance. The loss function used includes the MSE and an L_2 regularization term to prevent overfitting.

This approach allows the neural network to learn optimal design patterns and facilitates process automation, improving efficiency and reducing development time.

1.5 First case study: second-order SC $\Sigma\Delta$ Modulator

In this case study, the design of a second-order $\Sigma\Delta$ modulator implemented using *switched-capacitors* (SC) is addressed. This type of modulator is widely used due to its high precision and energy efficiency, making it ideal for digital audio and biomedical device applications.

The analysis focuses on optimizing the design variables that influence the modulator's performance, using neural networks to automate this process and reduce manual effort.

1.5.1 Dependence on the number of epochs

To evaluate the influence of the number of epochs, a neural network with 100 training epochs was used. The loss function and the mean absolute error (MAE) stabilize quickly, suggesting that 100 epochs are sufficient to achieve adequate convergence without overfitting.

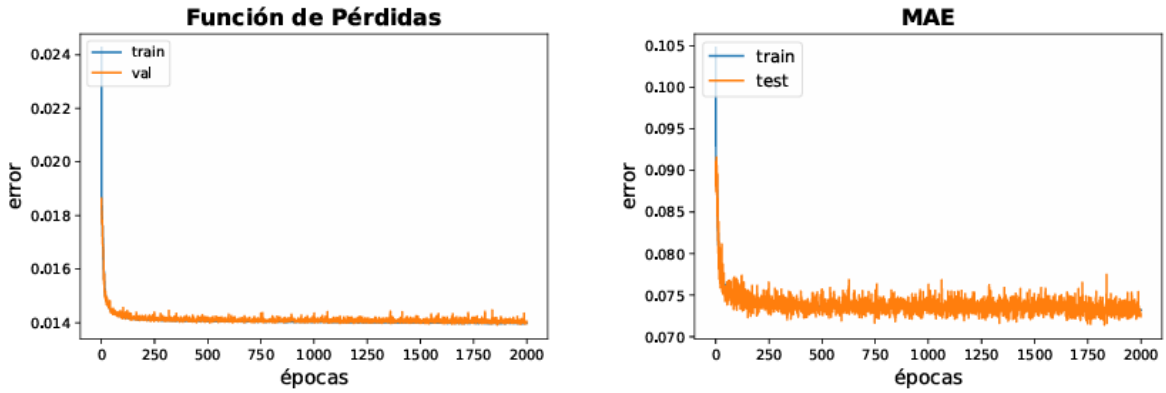

Figure 3: Dependence on the number of epochs

1.5.2 Dependence on batch size

The *batch* size significantly impacts network performance. Batch sizes between 64 and 8192 were evaluated, observing that a *batch* of 256 achieves an optimal balance between accuracy (MSE and MAE) and training time. Increasing the *batch* size reduces noise in updates but can hinder precise adaptation to the dataset.

Tamaño Batch	8192	4096	2048	1024	512	256	128	64
MSE	0.0169	0.0152	0.0148	0.0144	0.0143	0.0143	0.0142	0.0143
MAE	0.0853	0.0784	0.0776	0.0758	0.0753	0.0749	0.0743	0.0747

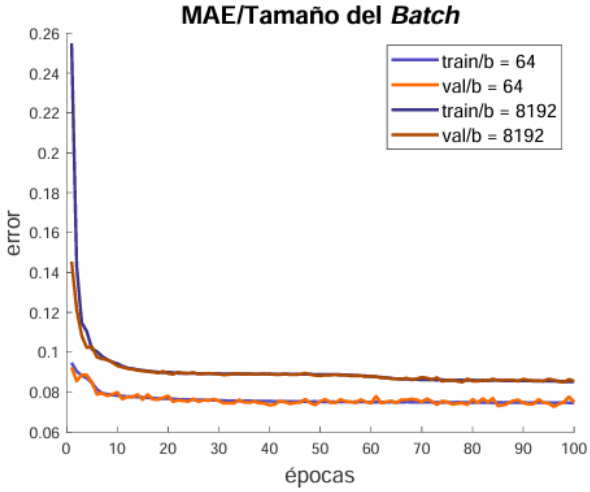

Figure 4: Dependence on batch size

1.5.3 Dependence on the augmentation factor

The augmentation factor (K) determines the extension of the training dataset by adding extra copies with modified specifications. Values of $K = 1$, 10, 40, and 400 were evaluated. Although dataset augmentation improves the network's ability to generalize beyond the original range, no significant improvements were observed in metrics such as mean squared error (MSE) and mean absolute error

(MAE). It was concluded that low values of K provide better training times without compromising accuracy.

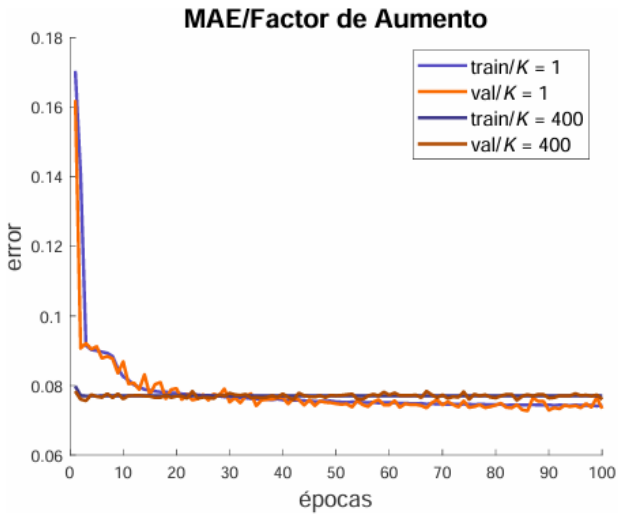

Figure 5: Dependence on the augmentation factor

1.5.4 Dependence on the number of neurons

The number of neurons in hidden layers directly affects the network's ability to learn complex patterns. Networks were evaluated using a scaling factor f applied to the initial number of neurons, where f varied from 1 to 10. Results showed that excessively increasing the number of neurons leads to higher errors due to issues like overfitting and numerical errors. Additionally, it was identified that the base network with fewer neurons already operated optimally, indicating that adding more neurons does not provide significant benefits and only increases computational resource requirements.

Neuronas	x1	x4	x7	x10
MSE	0.0142	0.0142	0.0580	0.1481
MAE	0.0743	0.0744	0.1607	0.3427

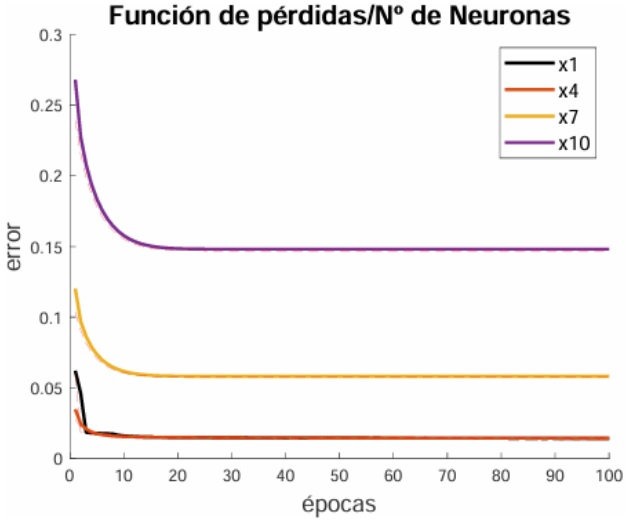

Figure 6: Dependence on the number of neurons

Why are worse results obtained? Poorer results in certain network configurations are attributed to two main factors:

- **Overfitting**: Occurs when the neural network is too complex for the dataset size, causing the network to memorize training data instead of generalizing correctly.
- **Numerical errors**: Excessively increasing the number of neurons and layers increases computational complexity, which can cause numerical precision issues during training.

These problems highlight the importance of finding a balance between network capacity and dataset size.

1.5.5 Dependence on the number of layers

The number of hidden layers affects the neural network's ability to model complex relationships between variables. Configurations with 1, 4, 7, and 10 hidden layers were evaluated:

- Networks with 4 layers showed better performance, reducing mean squared error (MSE) and mean absolute error (MAE).
- However, increasing the number of layers beyond 4 did not provide significant improvements and increased the risk of overfitting.

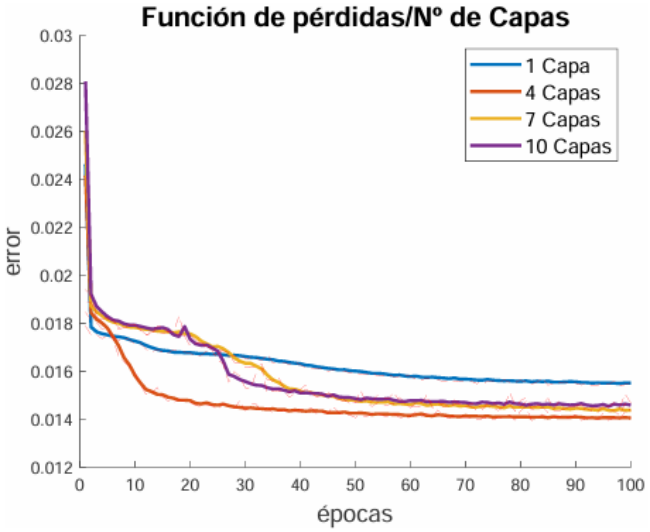

Figure 7: Dependence on the number of layers

It was concluded that a network with 4 hidden layers offers an optimal balance between accuracy and computational efficiency.

1.5.6 Dependence on the validation dataset size

The size of the validation dataset affects the model's ability to effectively evaluate its performance. Different validation dataset sizes were analyzed, observing that:

- A very small validation set may not be representative, leading to an unreliable model evaluation.
- On the other hand, an excessively large validation set reduces the amount of data available for training, negatively affecting the model's performance.

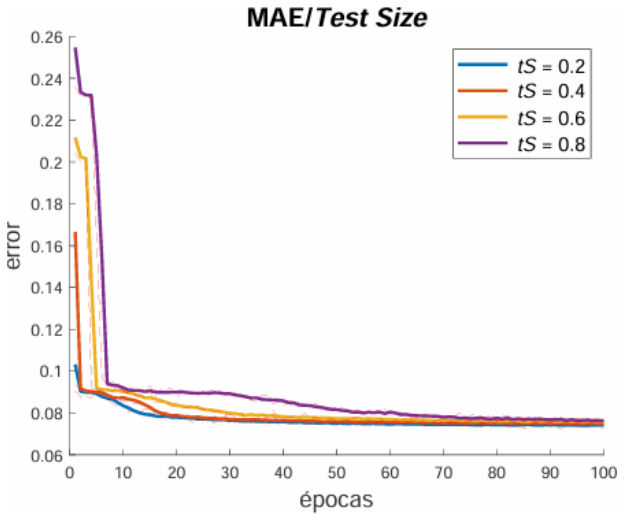

Figure 8: Dependence on the validation dataset size

The best balance was achieved with a validation set representing 20% of the total data, allowing for adequate evaluation without compromising training.

1.5.7 Summary of results

The analysis of the first case study showed that neural networks can effectively automate the design of second-order ΣΔ SC modulators. among the most relevant results:

- The optimal network configuration includes 4 hidden layers, a batch size of 256, and a validation dataset equivalent to 20% of the total.
- The obtained metrics validate the model, with a mean squared error (mse) of 0.0142 and a mean absolute error (mae) of 0.0743.
- Factors such as dataset size, number of neurons and layers, and batch size significantly impact model performance.

1.5.8 Validation of network predictions

To evaluate the accuracy and reliability of the trained neural networks, a **cross-validation** was performed using an independent dataset. results:

- A high correlation between network predictions and actual values confirms that the model generalizes correctly to unseen data.
- Prediction errors remain low across the entire range of specifications, demonstrating that the model is robust and applicable to different design configurations.
- Visualization of results showed a uniform error distribution, without significant biases towards high or low values.

These results reinforce the validity of the proposed approach and its ability to efficiently and accurately automate the design of ΣΔ modulators.

1.6 Second case study: second-order ΣΔ Gm-c Modulator

In this second case study, the design and analysis of a second-order ΣΔ modulator implemented using transconductances (*gm-c*, transconductance-capacitor) is addressed. This type of modulator is widely used in high-frequency applications, such as telecommunications and data conversion systems, due to its ability to operate at high speeds with relatively low power consumption.

The second-order ΣΔ Gm-c Modulator consists of:

- **Transconductors**: implement continuous-time integration operations, unlike the switched capacitors used in the first case study. Transconductors convert voltage signals into current signals.
- **Capacitors**: store charge, providing the necessary capacity to perform signal integration operations in the analog domain.
- **Quantizer**: as in the previous case, a one-bit quantizer is used, simplifying the design and allowing efficient noise shaping.

The modulator architecture includes a feedback loop that shifts quantization noise out of the band of interest, achieving a high signal-to-noise ratio (SNR) within the useful band.

The following design variables were analyzed:

- **Transconductance (g_m)**: influences the operating speed and dynamic range of the modulator.
- **Capacitance (c)**: affects the system's filtering properties.
- **Cutoff frequency (f_c)**: determined by the relationship between g_m and c, defining the modulator's operating range.

the specifications used to evaluate modulator performance were:

- Signal-to-noise ratio (SNR).
- Figure of merit (FOM).
- Power consumption (P).

To train the neural networks, a dataset was generated through parametric simulations of the modulator:

- A wide range of combinations of g_m, c, and f_c were considered.
- For each configuration, SNR, FOM, and P specifications were calculated using time and frequency domain simulations.
- The data was filtered to exclude invalid combinations or those not meeting design objectives.

This process resulted in a sufficiently representative dataset to capture the relationships between design variables and specifications.

A neural network was used with an architecture similar to that of the first case study, but adapted to the specific characteristics of the gm-c modulator. the main parameters were:

- **Inputs**: normalized design variables (g_m, c, and f_c).
- **Hidden layers**: three layers with 120, 240, and 60 neurons, designed to capture complex interactions between variables.
- **Outputs**: predictions of SNR, FOM, and power consumption specifications.

The training used a batch size of 256 and a learning rate optimized with the adam algorithm. multiple tests were conducted to adjust hyperparameters and avoid issues such as overfitting.

1.6.1 Summary of results

The obtained results demonstrate the effectiveness of the model:

- **Mean squared error (MSE)**: 0.0121, indicating high prediction accuracy.
- **Mean absolute error (MAE)**: 0.0625, reflecting good model generalization.
- **Correlation**: a high correlation between network predictions and actual values validates the model's ability to learn relevant patterns.

Additionally, predictions were consistent across the entire range of design variables, demonstrating the model's robustness.

1.7 Conclusions

- **Network structure**: it was confirmed that not overly complex networks, in terms of the number of neurons and layers, offer an optimal balance between performance and computational resources. These networks can be trained quickly on a personal computer and produce results comparable to those of more complex networks. In the validated training dataset points, the networks tended to predict a higher SNR than requested, which is a positive outcome.
- **Generalization ability**: Although networks perform worse when predicting for randomly generated points, it was observed that generating multiple predictions improves the probability of obtaining satisfactory results.
- **Comparison with other algorithms**: Neural networks stand out for their ability to directly predict the design variables needed to meet certain specifications, reducing prediction time once trained. However, dataset generation and training require considerable time, between 5 and 20 hours depending on the case study, surpassing the execution time of other algorithms, such as genetic ones. therefore, RNN are more advantageous when multiple designs are required.

2 Design automation of analog and mixed-signal circuits using neural networks - A tutorial brief

Date: January 28, 2025

2.1 Introduction

The development of technology has led modern electronic devices to rely predominantly on digital circuits due to their higher energy efficiency, programmability, and lower cost. Unlike digital circuits, whose design benefits from advanced electronic design automation (EDA) tools, the design of analog circuits still largely depends on empirical rules and prior experience.

Although various methodologies and EDA tools have been developed to automate the design of analog circuits, these strategies often employ optimization-based approaches, where a simulator evaluates the performance of possible solutions within the design space. In this context, artificial intelligence algorithms, especially artificial neural networks (ANNs), have begun to play an important role. ANNs can replace simulators or act as optimization engines, automating the design process and generating optimal solutions for new specifications.

This tutorial provides an overview of the application of ANN-based methods to the automated design of analog and mixed-signal circuits. Additionally, two case studies are presented:

1. System-level sizing of Sigma-Delta modulators.
2. Circuit-level design of operational transconductance amplifiers.

2.2 Optimization-based design methodology using ANNs and computational intelligence techniques

The design of analog and mixed-signal circuits follows a widely used hierarchical methodology known as the top-down/bottom-up design approach. This process divides the system into abstraction levels, typically: system, circuit, device, and physical. As system specifications are hierarchically passed from the top level to the lower levels, performance is verified in the reverse path, from the lower level to the top.

At each level of abstraction, the most suitable topology or architecture is selected. For example, in analog-to-digital converters (ADCs), this involves choosing the most appropriate conversion technique, such as pipeline or Sigma-Delta modulators, and within this family, selecting the optimal architecture. Once the topology is defined, the design variables necessary to meet the specifications at each level are determined.

This design process is typically carried out using an optimization engine that guides a simulator to explore the design space and find the optimal solution. The optimization techniques employed include genetic algorithms, simulated annealing, and multi-objective evolutionary methods. However, these approaches present challenges, such as long simulation times and multiple iterations required to converge to a solution.

2.2.1 Conventional optimization-based design method

At each level of abstraction in the design, the most suitable topology is selected, and the necessary design variables are determined to meet the specifications. For example, in Sigma-Delta modulators, typical system-level specifications include the Effective Number of Bits (ENOB) and bandwidth (BW),

which translate into circuit-level requirements such as direct current (DC) gain, gain-bandwidth product (GBW), and slew rate (SR). These system-level design variables become specifications for the lower levels.

This design process involves multiple iterations of simulation and verification at each level of abstraction. Electrical simulators, such as SPICE-based ones, are used to obtain high accuracy at the circuit level, while behavioral simulators, such as SIMSIDES, enable fast evaluation at the system level through simplified models that capture the circuit's main non-idealities.

2.2.2 ANN-driven optimzation-based design method

Although conventional optimization methods are effective, they present challenges such as the large number of required iterations and long simulation times. Neural networks can mitigate these problems by reducing the need for intensive simulations by:

- Selecting only the most promising solutions for evaluation.
- Estimating circuit performance objectives, such as behavior under design corners.
- Completely replacing the circuit simulator.

Additionally, ANNs can be used as reinforcement learning (RL) agents to optimize circuits, eliminating the need for a conventional optimization loop. This tutorial proposes the use of ANNs in a supervised approach, where they are trained to directly infer design parameters from specifications.

2.3 ANN-driven system-level design: aplication to $\Sigma\Delta$Ms

System-level design requires solving two fundamental problems:

1. Selecting an appropriate architecture within a family of alternatives based on a set of specifications.
2. Determining the optimal design variables that meet the specifications for the chosen architecture.

This approach combines neural networks for architecture classification and regression networks for inferring design variables.

2.3.1 Problem formulation and definition of the dataset

The first problem, architecture selection, is addressed using a trained classifier that maps the system performance metrics to a categorical variable representing the architecture. The second problem, a constrained optimization problem, utilizes regression neural networks trained on large datasets to infer the design variables that optimize performance.

For this approach, each entry in the dataset is organized as a triplet $\{\mathrm{C},\ \bar{s},\ \bar{v}\}$, where C is the categorical architecture, $\bar{s}$ are the system performance metrics, and $\bar{v}$ are the design variables resulting from behavioral simulations using SIMSIDES. The dataset is normalized to facilitate the convergence of the training.

2.3.2 Network architecture and model optimization

Various classifiers were evaluated, such as Discriminant Analysis (QDA, LDA), Support Vector Machines (SVM), and Random Forests, with Gradient Boosting achieving the highest score. For regression, dense neural networks were used, and their architecture was optimized using Neural Architecture

Search (NAS) techniques with Keras Tuner.

The accuracy of the networks was evaluated based on their ability to approximate the requested performance metrics. The networks were used to generate initial solutions, which were then slightly adjusted to improve the merit figure of merit (FOM). This process ensured that the solutions were optimal for the given specifications.

2.3.3 ΣΔM design examples

Four Sigma-Delta modulator architectures were considered as case studies:

- A 2nd-order ΣΔ modulator with switched capacitor (SC) technology.
- A 3rd-order cascaded ΣΔ modulator (2-1 SC).
- A 4th-order cascaded ΣΔ modulator (2-1-1 SC) with 3-bit quantization.
- A continuous 2nd-order ΣΔ modulator based on Gm-C integrators.

The specification vector included resolution (SNR), bandwidth (OSR), and power consumption (Pow). The behavioral models used considered the main circuit non-idealities, such as limited gain, amplifier noise, and limited bandwidth, but did not include effects like device mismatches or thermal noise.

The dataset was generated by evaluating over 200,000 random designs, with only those having SNR ¿ 50dB selected for training. The final dataset contained approximately 120,000 elements, which were divided into training, validation, and test data. The Gradient Boosting classifier achieved 93.7% accuracy on the validation set, making it the selected approach for this task.

2.3.4 Verification and comparison with other optimizers

The proposed methodology was validated by comparing the designs obtained with ANNs against conventional optimization algorithms, such as genetic algorithms and gradient descent. In a test set of 1000 points, the ANNs managed to generate centered designs that met the required specifications in at least 68.5% of cases, reaching up to 78.5% in specific architectures.

Moreover, the ANN-based methodology showed a significant improvement in CPU time, being at least 60 times faster than conventional optimizers. It also produced designs with lower power consumption and competitive metrics, achieving better values in most cases for the merit figure of merit (FOM), except in a particular case.

2.4 ANN-driven circuit-level design: application to OTAs

The circuit-level design focuses on obtaining the transistor dimensions and biases to meet a given set of specifications. In this case, artificial neural networks (ANNs) were employed to predict the optimal design variables for operational transconductance amplifiers (OTAs) based on the required specifications.

2.4.1 Preparing the circuit-design dataset

The dataset used to train the ANNs included data pairs in the form $\{\bar{s}_i, \bar{v}_i\}$, where $\bar{s}_i$ represents the circuit performance metrics (such as DC gain, gain bandwidth (GBW), current consumption (I_{DD}), among others) and $\bar{v}_i$ the design variables, such as transistor dimensions and multiplicity. Only well-designed circuits were included in the dataset to ensure optimal predictions.

Since design specifications are generally expressed as inequalities (e.g., $I_{DD} < 200\mu A$), the dataset was enriched through a data augmentation technique. For each design $\bar{v}_i$, multiple specification vectors $\bar{s}_k$ were generated that were worse (or equivalent) than the actual performance of the design $\bar{v}_i$. This allowed the ANNs to learn to generalize to a broader design space. Additionally, the dataset was normalized to improve training efficiency.

2.4.2 Defining and using the model

The neural network used consisted of a dense (fully-connected) architecture with three hidden layers. The dataset was divided into 80% for training and 20% for validation. Additional design points, generated independently, were used to evaluate the model's performance outside the training data.

The model predicted multiple solutions for each target specification $\bar{s}$. These predictions were generated from sets of random specifications that met the defined limits, thus increasing the coverage of the design space and reducing potential biases in the data. The obtained designs were evaluated through simulations to select the optimal solution using metrics such as the FOM or Pareto optimality to analyze trade-offs between relevant metrics.

2.4.3 OTA design example and results

As a case study, an operational transconductance amplifier (OTA) designed in 130 nm CMOS technology was considered. The dataset contained 16,600 points obtained from previous optimizations, extended with second-order polynomial features. The metrics used included DC gain, bandwidth (GBW), current consumption (I_{DD}), and phase margin (PM).

The neural network used had 14 input nodes (including polynomial features), three hidden layers with 120, 240, and 60 nodes, respectively, and an output layer with 12 nodes representing the transistor dimensions. The initial training on the original dataset took less than 15 minutes, followed by additional training on an augmented dataset of 700,000 samples, taking another 40 minutes.

The results showed that the model could predict transistor sizes that met the target specifications. For each set of specifications, the neural network generated 100 predictions with random deviations of up to 15% from the targets. In all cases, the obtained designs exceeded the required specifications or achieved FOMs greater than 1000. Even when the specifications were infeasible, the model proposed solutions that balanced the trade-offs between performance metrics.

2.5 Conclusion

The use of artificial neural networks (ANNs) for the automated design of analog and mixed-signal circuits and systems has been discussed in this tutorial. The presented methodology has been applied at the system level for the synthesis of Sigma-Delta modulators and at the circuit level for the optimization of operational transconductance amplifiers (OTAs). The results obtained are competitive with other optimization methods.

Although this technology is still in its early stages, the use of ANNs to automate the design of analog and mixed-signal circuits and systems can help bridge the gap between electronic design automation (EDA) tools for digital and analog circuits.

3 On the use of artificial neural networks for the automated high-level design of Sigma-Delta modulators

Date: January 29, 2025

3.1 Introduction

Sigma-Delta modulators are one of the most effective techniques for implementing analog-to-digital converters (ADCs) in a wide range of applications, including instrumentation, biomedical devices, automotive sensors, and communications. Although they offer significant advantages in terms of robustness and efficiency, designing high-performance ΣΔMs is complex and requires expertise in a hierarchical top-down/bottom-up design flow that spans from the system level to the chip level.

The main challenge in design lies in finding the appropriate architecture for a given set of specifications and mapping these specifications to electrical parameters at the circuit level. In recent years, design methodologies and automation tools (EDA) have been developed to optimize system-level tasks such as architecture selection, loop filter design, behavioral modeling, and sizing. The most common approach is based on optimized synthesis methods, where an optimizer works with a simulator to find the optimal design.

Recently, artificial intelligence algorithms, such as artificial neural networks, have begun to play a key role in automating analog circuit design, allowing simulators to be replaced or acting as optimization engines. These networks can automate system sizing for new specifications, significantly reducing the time and computational resources required.

This paper presents a methodology based on ANNs for automated high-level design of ΣΔMs, combining architecture classification and regression to infer design variables. The methodology is valid for both single-loop and cascaded architectures, with single or multi-bit quantization, and continuous-time or discrete-time circuit techniques. The results demonstrate significant improvements in terms of CPU time and performance compared to other optimization methods.

3.2 Background and prior art on optimization-based synthesis of ΣΔMs

The design methodology of ΣΔ modulators follows a hierarchical top-down and bottom-up approach. In this divide-and-conquer strategy, a ΣΔ modulator is divided into several abstraction levels (system level, building block level, circuit level, and device/physical level), ensuring that at each level, a design process is carried out to hierarchically allocate system specifications.

The design process of ΣΔ modulators involves selecting the best architecture that meets the ENOB and bandwidth specifications with the lowest possible power consumption. For this purpose, ideal equations are used to estimate key parameters such as the Oversampling Ratio (OSR), the loop filter order (L), and the number of bits of the quantizer (B). More accurate models are later applied using tools like MATLAB's ΣΔ Toolbox.

Once the modulator architecture is defined, circuit non-idealities are analyzed to specify the main building blocks, such as amplifiers and comparators. For this, accurate and efficient simulators like SIMSIDES, a MATLAB/SIMULINK-based simulator, allow rapid and precise modeling of circuit non-idealities.

To optimize the design, techniques such as behavioral modeling and the lifting method are used, the latter aimed at continuous-time ΣΔ modulators (CT-ΣΔ). Tools like the Sigma-Delta Synthesis Tool automate the design, enabling the optimization of loop filter parameters and the Signal Transfer

Function (STF).

Automated design of ΣΔ modulators relies on optimization engines that explore the multidimensional design space. However, this work proposes replacing these systems with artificial neural networks (ANNs), which learn to map specifications to design parameters from extensive datasets. Although this article does not address the use of ANNs for loop filter coefficient sizing, this application could be explored in future work.

3.3 Proposed methodology

3.3.1 Problem definition

The high-level design of a ΣΔ modulator requires solving two key problems:

- **Architecture selection**: A suitable topology must be chosen from a set of alternatives $\{A_j\}$, considering specifications such as ENOB (effective number of bits, the actual resolution of an ADC), BW (bandwidth, the range of input signal frequencies that the system can process without significant degradation), SNR (signal-to-noise ratio), SNDR (signal-to-noise and distortion ratio), THD (total harmonic distortion, the amount of unwanted harmonic energy in a signal), and FOM (figure of merit, a factor used to compare designs), along with additional constraints imposed by the system.
- **Estimation of design variables**: For the selected architecture, the design variables $\bar{\epsilon}(A_j)$ must be determined to optimize the given specifications.

3.3.2 Proposed solution

To address the architecture selection problem, a trained classifier C is used to assign a topology A_j based on system metrics $\bar{0}_i$. Subsequently, a regression-type neural network (RNN) infers the corresponding design variables $\bar{\epsilon}_k$:

$$A_j = C(\bar{0}_i); \quad \bar{\epsilon}_k = RNN(\bar{0}_i, A_j) \tag{5}$$

The use of pre-trained classifiers and neural networks significantly reduces inference time, achieving estimations in less than 50 ms. However, the quality of the results directly depends on the accuracy of the classifier and the neural network, as well as the quality of the dataset used for training.

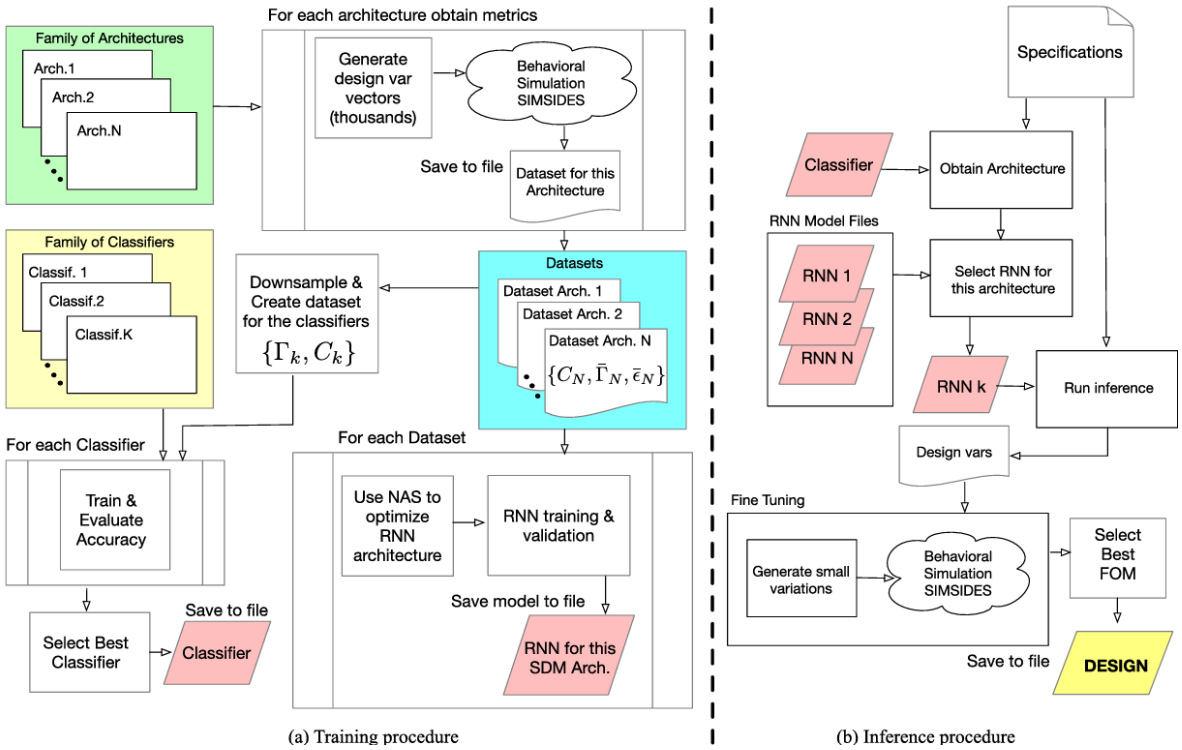

Figure 9: Complete methodology flow. The process is divided into two main phases: training and inference.

Training procedure: The objective is to train models capable of predicting the design parameters of $\Sigma\Delta$ modulators.

- **Architecture selection**: A set of possible architectures is considered, thousands of variable vectors are generated, evaluated to obtain performance metrics, and results are stored in datasets organized by architecture.
- **Classifier training**: A reduced database is built from simulations and used to train multiple classifiers. The best model is selected and later used in the inference phase to choose the appropriate architecture based on design specifications.
- **Training of recurrent neural networks** (RNNs): For each dataset, the RNN architecture is optimized using NAS (Neural Architecture Search). The RNNs are trained and validated with simulated data, and a trained RNN is stored for each $\Sigma\Delta$ modulator architecture.

Inference procedure: The trained system is applied to generate an optimal design based on given specifications.

- **Architecture selection**: The desired design specifications are received, and the classifier predicts the most suitable $\Sigma\Delta$ architecture.
- **Inference with neural networks**: The RNN corresponding to the chosen architecture is selected, and inference is performed to obtain the design variable values.
- **Fine-tuning and selection of the best design**: Small variations in the obtained parameters are generated, and simulations are run to refine the results. The best solution is selected based on FOM, and the final design is stored as the process output.

3.3.3 Designing the dataset

Our proposal is to structure each dataset entry as a triplet of the form $\{C_i, \bar{0}_i, \bar{\epsilon}_i\}$, where:

- C_i is a categorical variable defining the modulator architecture.
- $\bar{0}_i$ is a vector of system performance metrics, i.e., a point in the specification space.
- $\bar{\epsilon}_i$ is a vector of design variables, a point in the design variable space, obtained from behavioral simulations with SIMSIDES.

The specification vector $\bar{0}_i$ must be common to all architectures in the dataset so that the classifier can operate on it. However, the design variables $\bar{\epsilon}_i$ vary depending on the architecture. To address this, two approaches exist:

1. A single dataset with all architectures, using a unified design vector that includes all necessary variables, marking as N/A those that do not apply to a specific architecture.
2. Separate datasets for each architecture, optimizing each regression neural network independently for its corresponding architecture.

The second approach is more scalable and efficient, as it allows training neural networks specific to each architecture without needing to reformulate the entire dataset when adding new topologies.

3.3.4 Considerations on the classifier definition

Selecting an appropriate classifier architecture is crucial for model success. There is no single optimal technique, as the best choice depends on the nature of the dataset and the statistical properties of the information. Factors such as class imbalance, feature correlation, dimensionality, and noise influence this decision.

To avoid classification bias issues, we use a balanced subset of the dataset, ensuring an equitable representation of all classes. A procedure is implemented to explore various classification techniques, selecting the best option based on the obtained confusion matrix. The classifiers considered include:

- Linear and quadratic discriminant analysis.
- Support vector machines (SVM) with different kernels.
- Gaussian and multinomial naive Bayes.
- Decision trees and random forests.
- Gradient boosting classifiers.
- Neural networks.

This approach allows selecting the best classifier based on the specific training and validation data, maximizing the accuracy of optimal architecture prediction.

3.3.5 Network architecture search and model optimization

The foundation of our solution is a multivariable regression neural network optimized for each $\Sigma\Delta$ modulator architecture. This network maps system metrics (specifications) to high-level design variables, playing the role of the optimizer in the conventional approach.

However, manually defining a network architecture that minimizes regression error is a tedious and time-consuming process. To overcome this, we have integrated Neural Architecture Search (NAS) techniques into our design framework. NAS automatically explores the hyperparameter space and finds an optimal network architecture using the Keras Tuner API.

Our approach first defines a parameterized network template and a search space for the following hyperparameters:

- Number of neurons per layer: [32, 64] with increments of 4.
- Number of additional layers after concatenation: [1, 6] with increments of 1.
- Activation function: ReLU or hyperbolic tangent (tanh).
- Dropout layers: enabled or disabled.
- Optimization algorithm: Adam, SGD, RMSprop, Adadelta.

The search algorithm runs until all combinations in the search space are exhausted or an early stopping condition is met. Subsequently, the best-found architecture is fine-tuned with a training and validation set in an 80%-20% split.

3.3.6 Cross-validation and result improvement

The mean squared error (MSE) is used to quantify the accuracy of neural networks. However, due to the nonlinear nature of $\Sigma\Delta$ modulators, this error does not provide a clear insight into the fidelity of the obtained design parameters relative to the required specifications. To address this issue, we integrate an additional phase of cross-validation and result refinement.

This phase consists of generating a small number of random variations of the initial solution provided by the network, with slight adjustments to its components:

$$DV_{n,a} = DV_n(1 + \alpha_{n,a}), \quad \alpha_{n,a} \sim U(-r, r) \tag{6}$$

where r represents an adjustment percentage and U is a uniform distribution. These variations are evaluated in a behavioral simulator to select the solution with the best Figure of Merit (FOM).

This approach combines the inference speed of neural networks with the accuracy of simulation, allowing the discovery of better-performing design configurations without a high computational cost.

3.4 Case study

The four $\Sigma\Delta$ modulator architectures selected as case studies are:

1. 2nd-order SC modulator.
2. 3rd-order cascade 2-1 SC modulator.
3. 4th-order cascade 2-1-1 SC modulator with 3-bit quantization.
4. 2nd-order Gm-C modulator with 3-level quantization.

To train the neural networks, a dataset was generated through simulations in SIMSIDES. Five values of the oversampling ratio were considered ($OSR = 32, 64, 128, 256, 512$), and the design variable spaces were explored for each architecture:

- For the 2nd-order Gm-C and 3rd-order 2-1 SC modulators, design variables were generated through nested loops (systematic sweep).
- For the other two cases (4th-order 2-1-1 SC and 2nd-order SC), design variables were generated randomly within predetermined ranges.

The data generation process included more than 200,000 simulations. However, due to randomness in design variable exploration, not all simulations produced useful results. Therefore, designs with a signal-to-noise ratio (SNR) below 50 dB were filtered out, reducing the final dataset to approximately 120,000 data points. Each dataset entry was structured as a triplet:

$$\{C_i, \Theta_i, \epsilon_i\} \tag{7}$$

where:

- C_i: modulator architecture (categorical variable).
- Θ_i: specification values (SNR, OSR, power consumption).
- ϵ_i: vector of design variables obtained from simulation.

3.4.1 Training and validating results

The training of ANNs involved two phases:

1. **Architecture classification**: A network was used to infer the most suitable modulator architecture based on a set of specifications.
2. **Regression for design variables**: A separate ANN was trained for each architecture to predict the corresponding design variables given a set of specifications.

Classifier training: To avoid class imbalance issues, the dataset was balanced so that each architecture had the same number of entries. Various classification approaches mentioned earlier were evaluated. The best-performing classifier was the **Gradient Boosting Classifier**, with the best confusion matrix.

Regression network training: For each architecture, a neural network was designed by exploring different hyperparameters. Once the best configuration for each network was determined, training was conducted (80% training set and 20% validation set). The metric used was MSE.

Validation and final adjustment: 1,000 validation data pairs were randomly selected, and the relative error in the obtained specifications compared to the desired ones was calculated. The neural networks achieved minimal deviation in design variable prediction, ensuring that the required specifications were met with high accuracy. To further improve results, a local search process was implemented around the obtained solution, based on simulating small random variations in design variables and selecting the one that produced the best Schreier Figure of Merit (FOM).

3.5 Discussion

The performance of this methodology was compared with three traditional optimizers: genetic algorithms, gradient descent, and positive basis Np1.

- **Computational time efficiency**: The ANN-based solution was 60 times faster.
- **Solution quality**: Although ANNs do not have direct optimization capability, the fine-tuning process with local search enabled the attainment of solutions with lower power consumption and better Schreier Figure of Merit (FOM_S) in most cases.