Deep Neural Networks for Behavioral Modeling of Analog Integrated Circuits

André Amaral, Nuno Lourenço, Ricardo Martins, Nuno Horta
Instituto de Telecomunicações, Lisboa, Portugal
Instituto Superior Técnico – Universidade de Lisboa, Lisboa, Portugal
{andre.c.amaral; nlourenco; ricmartins; nuno.horta}@tecnico.ulisboa.pt

***Abstract* – This paper is integrated into the Electronic Design Automation (EDA) area, and its main objective is to automatically create behavioral models of analog circuits from simulation data. To achieve the proposed goal, Artificial Neural Networks (ANNs) are used to model the circuit, speeding up the simulations and providing residual error results when compared with an off-the-shelf simulator. Since the pair input-output is generated through SPICE simulation, the type of learning is supervised. The work proposes to model the circuit behavior using Recurrent Neural Networks (RNN), more specifically a "Long Short Term Memory" ANN (LSTM) and a MultiLayer Perceptron (MLP) with delay lines. As a novelty, this work proposes an approach that can model the circuit behavior for different circuit sizes. The proposed method was applied to a set of amplifiers and the obtained results show the effectiveness of the LSTM and MLP networks in the behavioral modeling of analog circuits. It also presents a generator script capable of converting the Python ANN to Verilog-A language, which was used to convert the MLP model from Python to this Hardware Description Language (HDL). So, an ANN in Verilog-A is obtained, ready to be integrated in complex circuits, 5 times faster than the simulation at the transistor level.**

Index Terms – Analog Circuits, Long Short Term Memory, Behavior Modeling, Electronic Design Automation, PyTorch.

I. Introduction

In a "Mixed-Signal (MS) System-On-Ship (SoC)" the analog circuitry, which occupies a lower area compared to the digital circuitry [1], is the bridge between the digital circuits and the physical devices, allowing, for example, sensing the system inputs, to convert the signal between analog and digital and regulate or provide power to the circuit. Despite Integrated Circuits (IC) not having an "eye catch" development, circuits are getting more complex, power-efficient, and with reduced sizes, which shows the improvements in this area. The slow development is related to the lack of design automation support since it requires exhaustive knowledge and is less systematic.

Recently, Machine Learning (ML) is starting to trace its path in this area, allowing to create models of the circuit's behavior that mimics their functional behavior speeding up the simulation time.. With novel uses of machine learning [2], particularly Artificial Neural Networks (ANN) and Deep Learning (DL), it is possible to close the accuracy gap between Analog & Mixed-Signal (AMS) ICs' behavioral models and their corresponding spice-level (pre-and post-layout) models.

In this work, through a single Long Short Term Memory (LSTM) neural network, is possible, not only to model the circuit behavior with accuracy but also doing it for different device sizes which is an innovation in this research area. A state-of-the-art MLP with two delay lines, which also captures the circuit's behavior for multiple sizings, is also built. The MLP model is converted to Verilog-A using a generator script which is build to aid in this task since this HDL has a lack of support.

This paper is organized as follows: first, it is done a review of the state-of-the-art, and then it is presented the proposed architectures, explaining all the steps taken to achieve the results. Finally, the obtained results are presented, and the conclusions are drawn.

II. Related Work

The behavioral modeling of analog circuits or devices using ML techniques, such as ANNs and Support Vector Machines (SVMs), has been covered in many publications. Also, the conversion of the ANNs to Verilog-A has been explored in recent publications.

In [3], the behavior of a switched-capacitor three-stage cross-coupled charge pump circuit is modeled. First, the data is obtained by circuit simulation, and the inputs (current between nodes, clock time, and voltage) and the correspondent outputs are preprocessed, leading to a balanced dataset. A Multilayer Perceptron (MLP), with 3 hidden layers, is built to find the charge amount and implement a specific charge transfer function. The resulting model deviates a maximum of 9.6% from the circuit behavior and a reduction of 7 seconds in the simulation time.

In [4], a model is created to analyze the power consumption of a circuit transient since the manually written behavior models cannot capture the power consumption at the transistor level. To prove the theory, the authors used a low relaxation oscillator. The inputs (frequency, time clock, and enable) are obtained from circuit simulation in SPICE with a sample period of 10ns, and only the equidistant data will be kept. Once the data is extracted, a Time-Delay Neural Network (TDNN), with 2 hidden layers and 10 steps of delay, is taught to mimic the circuit's behavior. The results show that the power consumption curve differed with an error of 2.7% between the use of the model and the simulation of the circuit. The simulation time is reduced by 95% in relation to the SPICE simulations.

In [5], a Compositional Neural Network (CompNN) is built using a nonlinear autoregressive Neural Network with Exogenous Input (NARX) and a TDNN. Each feature has a small-sized NARX allowing for an observable model where the behavior of each feature can be analyzed separately. The output of each NARX is the input of the TDNN, which computes the

final output of the MIMO circuit. As a study case, a CMOS Bandgap voltage Reference circuit (BGR) is modeled. First, each feature passes through a respective small-size NARX and finally, a TDNN, with 1 hidden layer, merges the outputs of each small-size NARX and generates the model output. The results show an error-rate of 5% for a 1-V Output and a decrease of the simulation time by a factor of 17.

In [6], [7], and [8], the behavior of an oscillator, an oscillator with a buffer and an oscillator with a voltage controller, respectively, is mimicked, using a Feed Forward Neural Network (FFNN) which has 2 hidden layers with 20 and 10 hidden neurons. The model also includes a Periodic Unit which simulates the oscillator transient. Regarding the results this approach speeded up the simulations time, reducing it by 96%.

In [9], an SVM is used for regression problems to model analog circuits. To prevent overfitting and find the best hyperparameter's value (C-penalty degree for regression errors, γ and ε-RBF parameters) cross-validation and Grid Search is used. As the case study, a Source Coupled Fet Logic (SCFL) Buffer Cell is used to approximate the voltage and current output curves. The results are a mean squared error (MSE) of $2.62 * 10^{-14}\, A^2$ for the current and $3.20 * 10^{-6}\, V^2$ for the voltage.

In [10], the SVM is used to classify the design region as infeasible or not and prune the ones that are not, excluding a large portion of the design space. The generated feasibility classifier can work with the performance macro models in analog synthesis, providing the designer with a good starting point.

In [11] is proposed the translation of an Recurrent Neural Network (RNN) behavior model to Verilog-A in order to make a modified nodal analysis of the model. For that purpose, the difference equations of the RNN were converted to differential equations and the model evaluation was performed using different time steps.

III. Proposed Architecture

As an innovation, this work uses a single LSTM model that can modelate the behavior of the circuit for different circuit sizes. It is also presented an MLP with two delay lines, the first with one sample of delay and the second with two samples of delay, to model the circuit behavior allowing to have a comparison term. It is important to mention that the models are called parametric models since they account with the design parameters of the circuit.

In this section the implementation steps of the proposed models are presented with detail. It is also presented the VerilogA conversion of the MLP model and the generator script developed and used for the conversion. It is relevant to mention that the models are build using the open-source machine learning framework PyTorch [12].

A. *Data Acquisition*

As with any deep learning model, data is a relevant part of it, allowing to make accurate models. It is also extremely important to extract a good set of data, because if a model receive a poor dataset, it will underperform, no matter how complex the model is. For this work, the samples, extracted from the Spectre Simulator, are obtained from the output of a front-end optimization. For each device dimension is used nine datasets with two input waves and two output waves, the first four datasets the amplitude varies along the time and in the last 5 datasets the frequency changes keeping the amplitude constant. For this model was considered 93 different sizes of the circuit devices.

B. *Data Preprocessing*

To build a proper dataset for the LSTM model, the nine datasets containing the waves are opened and split into features and labels for each dimension. In order to keep the temporality of the data for each dataset, a sliding window is applied to obtain tuples of features and the corresponding label. At this point, the nine preprocessed datasets features are concatenated row-wise, as well as the labels obtaining the X and Y datasets, respectively. In Table 1, it is possible to see a short table with the data of the wave dataset.

Table 1 – Wave Dataset

V_{ip}	V_{op}	Out +	Out –
2.50000e-01	2.50000e-01	8.70962e-02	8.70962e-02
2.50007e-01	2.49993e-01	8.65948e-02	8.72738e-02
2.50010e-01	2.49990e-01	8.68060e-02	8.77894e-02
2.50429e-01	2.49571e-01	7.45603e-02	9.90171e-02
2.49505e-01	2.50495e-01	1.00598e-01	7.28204e-02

It is calculated, for all the sizes of the devices, the respective polynomial features, of order 2, to extract more relevant relations between the sizings and to have more features to make easier the learning phase. Those new features are concatenated column-wise with the X dataset. Finaly, features and labels are scaled using the MinMaxScaler to make each feature vary in a given range. The block diagram of the data-preprocessing is shown in Figure 1.

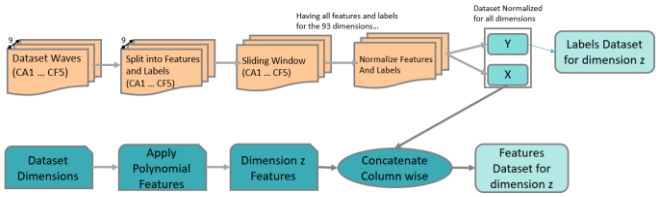

Figure 1 - Data Preprocessing Scheme for LSTM Model

To create a proper dataset for the MLP with two delay lines the data used is the same as the LSTM model, the data-preprocessing suffer an alteration where the features, after being normalized suffer a shift in the time, in this case one and two shifts in order to have 2 delay lines. Figure 2 is representative of that process.

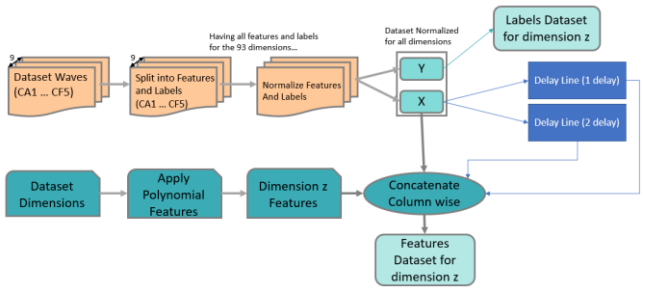

Figure 2 - Data Preprocessing Scheme for MLP Model

C. *Deep Learning Model's Structure*

As it is firstly proposed, a Sequence to Vector LSTM with is used, where given n samples, it predicts a unique output, and, regarding to the states, it is stateful since after each sequence prediction, the hidden and cell states keep their values which serves as the hidden and cell state for the next sequence.

The LSTM model created for this work splits the dataset into waves and circuit sizes. The sub-dataset, which has the waves, are the input of the LSTM layer. The output of the LSTM layer is reshaped and concatenated with the dimension's dataset column-wise. This concatenation is the input of a fully connected layer that has ELU as its activation function. In order to prevent overfitting, a dropout layer is introduced with the probability of 0.1. The output of this first layer is the input of another fully connected layer with a linear activation function. In Table 2 is possible to see the summary of the LSTM's hyperparameters. It is important to mention that to obtain those values, a tuning of the hyperparameters of the model was performed to seek the configuration that gives better performance.

Table 2 – LSTM Model Summary

Hyperparameter	Value Used
LSTM Layer	1 Layer (6 nodes)
Hidden Layers	2 Layers (450, 250 nodes)
Output Layer	1 Layer (2 nodes)
Activation Function	ELU – Hidden
Optimizer	AdamW
Learning Rate Scheduler	Factor = 0.7, patience = 2
Regularizer	Dropout (p = 0.1), Weight Decay (0.01)
Loss function	MSE
Batch Size	10000 (1% of total data size)
Gradient Clipping	0.25

Relatively to the MLP model with two delay lines the input waves flow through the first fully connected layer with activation function ELU and then the output is concatenated with the circuit sizes. Thus, it flows through the second fully connected layer which has ELU as activation function, which has a dropout layer with probability of 0.1 to prevent the overfitting effect, next it serves as the input of the third fully connected with Sigmoid as activation function, the output flows through the fourth fully connected layer with ELU as activation function. Finally, since this is a regression problem, the fifth fully connected layer has Linear as the activation function. Table 3 shows in detail the sum up of the MLP with two delay lines model.

Table 3 - MLP Model Summary

Hyperparameter	Value Used
Input Layer	1 Layer (6 nodes)
Hidden Layers	3 Layers (150, 35, 25 nodes)
Output Layer	1 Layer (2 nodes)
Activation Function	ELU & Sigmoid & Linear
Optimizer	AdamW
Regularizer	Dropout (p = 0.1), Weight Decay (0.01)
Loss function	MSE
Batch Size	10000 (1% total data size)
Gradient Clipping	1

D. *Adding Noise to the LSTM and MLP Models*

As expected, the ANN is immune to the circuit noise, so it needs to be studied and added to the predictions. There are different approaches, ones more sophisticated than others such as using variational Auto-Encoders to learn the noise behavior. In this work in the train phase the error between the predictions and the labels is computed and then the mean and standard deviation are obtained. To the model prediction a random gaussian noise with that mean and standard deviation is added, allowing to mimic the noise effect on the predicted data. The expression (1), (2) and (3) explains mathematically the noise addition process where the final output is the Noisy Predictions.

$$Error = Predictions - True\ Labels \quad (1)$$

$$Noise = Gaussian(mean, standard\_deviation) \quad (2)$$

$$Noisy\ Predictions = Prediction - Noise \quad (3)$$

In Figure 3 is a scheme of all the LSTM model that summarizes the model flow.

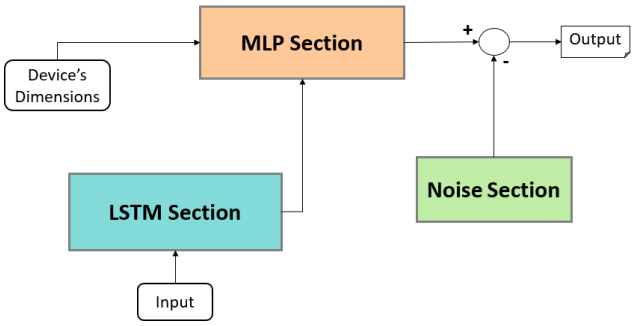

Figure 3 - Model Scheme

Figure 4 presents the scheme for the MLP model with two delay lines, summarizing the model flow.

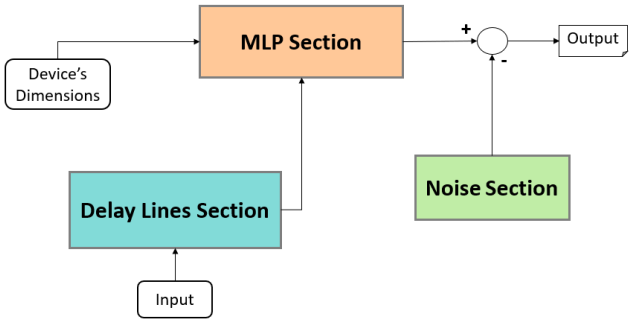

Figure 4 - MLP Model Scheme

E. *Verilog-A MLP Conversion*

To be possible to implement the model in complex front-end circuits in order to optimize the simulation time, it is necessary to convert it to an Hardware Description Language (HDL), in this case Verilog-A. This HDL has lack of support so, in order to automatize the process a generator script was created which automatically converts the Python code of a Fully Connected Layer, activation functions and necessary elements to perform delay lines and concatenations. This generator script brings innovation to this research field allowing to automate the conversion of the ANNs from Python to Verilog-A.

Briefly the forward path of the fully connected layer is given by the equation (4), where i is the number of neurons at the output of a fully connected layer and a is the output of the fully connected layer .

$$a[i] = \sum_{j=1}^{n} V(x[j]) * W_j + a[i] \qquad (4)$$

The equation (5) clarifies how the address to the output the correspondent value adding the bias.

$$\sum_{i=1}^{n} V(y[i]) = \sum_{i=1}^{n} a[i] + b[i] \qquad (5)$$

The activation functions, ELU and Sigmoid were created following its own expressions. The ELU Verilog-A code is presented in Figure 5. Since the neural network require normalized data, a normalizer was created using the MinMaxScaler approach with the respective minimum and maximum value used in Python scaling. The same happens for the de-scaler.

```
module ELU(n_deact, n_act);

  input n_deact;
  output n_act;
  electrical n_deact, n_act;
  real result;
  real p_num;
  parameter real alpha = 1;

  analog begin
      p_num = V(n_deact);
      if(p_num > 0)
          result = p_num;
      else
          result = alpha * (exp(p_num) - 1);
      V(n_act) <+ result;
  end

endmodule
```

Figure 5 - ELU in Verilog-A Language

It is important to mention that all the Verilog-A code is automatically generated by the script build for the purpose of transcript Python ANNs to Verilog-A Language. The generator script takes as inputs the weight and bias matrices, the input dimension and the output dimension. The script outputs the Verilog-A code for that respective fully connected layer. In Figure 6 is possible to see the function used to perform that task.

```
def FC_HiddenLayer_to_VerilogA(W, B, input_dim, output_dim):
    """
    Implements the code for a Fully Connected Layer in Verilog-A

    :param W: Weight matrix for a specific layer.
    :type W: float

    :param B: Bias matrix for a specific layer.
    :type B: float

    :param input_dim: Number of neurons in the input layer.
    :type input_dim: float

    :param output_dim: Number of neurons in the output layer.
    :type output_dim: float
    """
```

Figure 6 - Function Generator Script

Once having all the modules created the schematic is generated and the simulation is performed. It is also relevant to present a top architecture to present to the designer what inputs and outputs are needed to make this module work, so Figure 7 presents that top architecture.

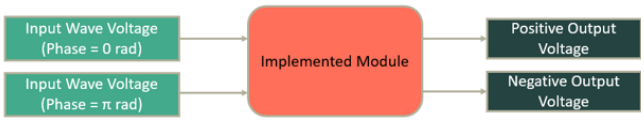

Figure 7 - Top Architecture for Designers

In the next section the results for the Python models and the Verilog-A model are presented.

IV. Case Study and Results

To validate the functionality of the proposed architectures, an OTA was used, obtaining the input and output data from the Spectre Simulation and it was preprocessed according to what was mentioned previously. Figure 8 presents the test bench where the data was extracted. Figure 9 is the circuit referent to the OTA whose the behavior will be mimicked through a LSTM model, a MLP model and the last model converted to Verilog-A.

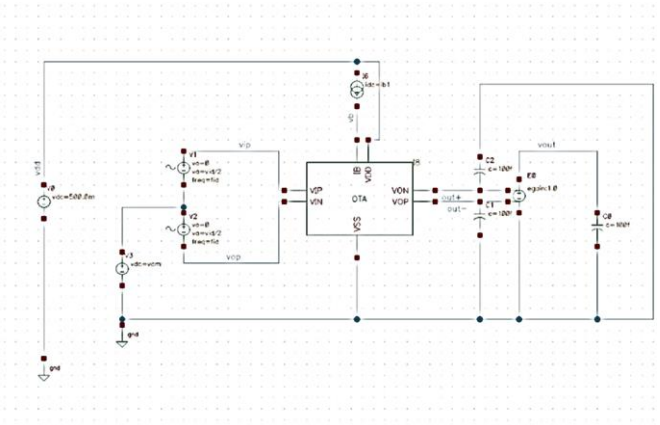

Figure 8 - Test Bench

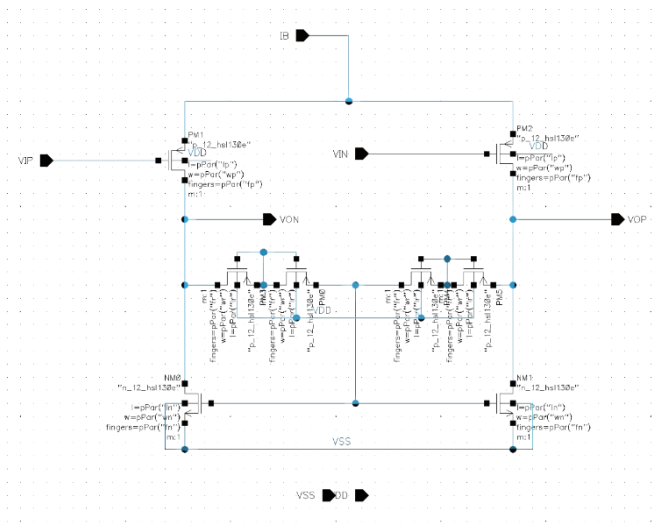

Figure 9 - Operational Transconductance Amplifier

A. *LSTM Model Results*

Training the model using the hyperparameters presents in the previous section (Table 2) leads the model to learn the data behavior, results that can be seen in Figure 10 and Figure 11. Those results do not infer how the model performs on unseen data but gives the insurance that the model can learn the data with accuracy.

In the following plots, the blue line represents the label and the orange one represents the predictions made by the model.

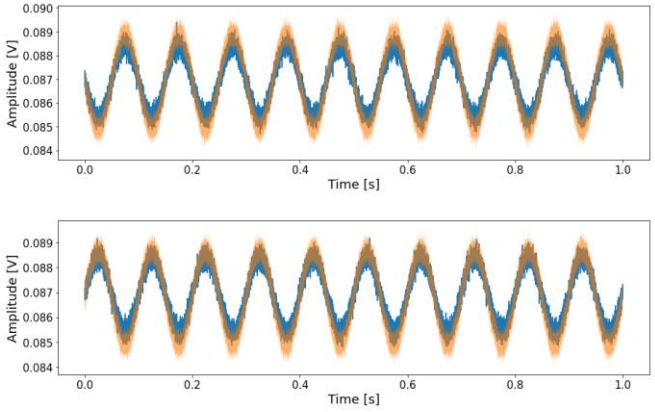

Figure 10 - Train Results 1 LSTM

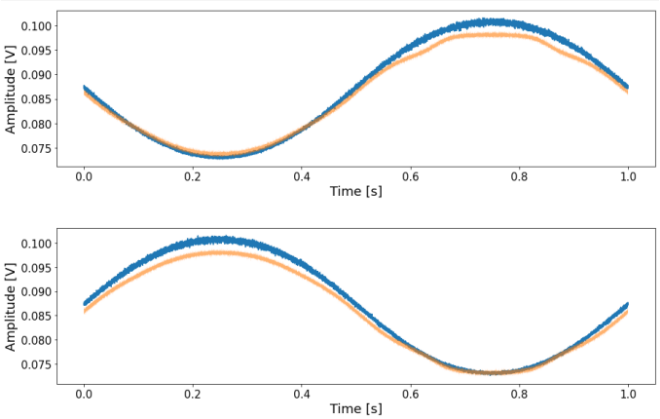

Figure 11 - Train Results 2 LSTM

It is also worth to mention that those results (present in Figure 10 and Figure 11) have an high level of accuracy, having a MSE of $2.3348 * 10^{-7}\, V^2$ and $2.7162 * 10^{-6}\, V^2$, respectively. The train results also present the gaussian noise added to the prediction.

To know if the model performs with higher accuracy with unseen data, two new datasets are created and given to the model. Figure 12 and Figure 13 presents the results for the test.

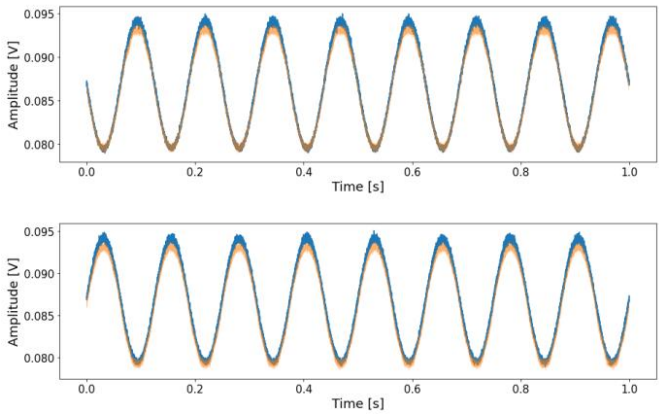

Figure 12 - Test Results 1 LSTM

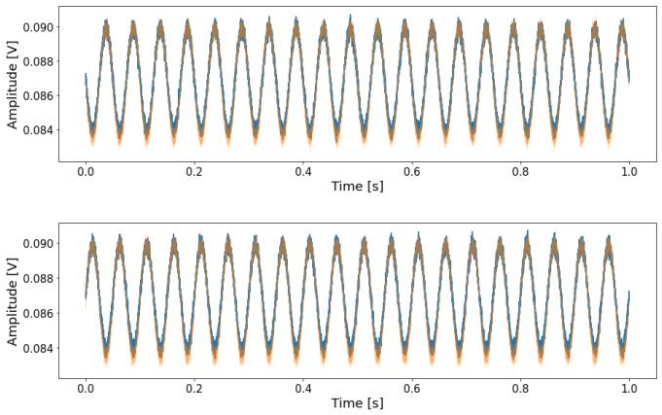

Figure 13 - Test Results 2 LSTM

For the test results1 a MSE of $4.9505 * 10^{-7}\, V^2$ was obtained and for the test results2 $1.7136 * 10^{-7}\, V^2$ was reached, proving the capability of the model to generalize to unseen data.

The generated noise has, the mean value $-4.3837 * 10^{-5}$ and the standard deviation value is $2.5118 * 10^{-4}$.

Finaly, the circuit where the point corresponds to the dimension leads the model to performed better than if the circuit have a different point and dimension. This behavior is captured in the fully connected layers where the dimension enters. In Table 4 is presented the MSE produced by the model for random points and random circuit sizes.

Table 4 – Circuit Sizes Results LSTM

Size	MSE Point 0	MSE Point 53	MSE Point 90
0	$\mathbf{2.2676 * 10^{-6}}$	$1.5458 * 10^{-6}$	$2.2509 * 10^{-6}$
25	$2.2685 * 10^{-6}$	$1.5446 * 10^{-6}$	$2.2535 * 10^{-6}$
33	$2.2694 * 10^{-6}$	$1.5442 * 10^{-6}$	$2.2521 * 10^{-6}$
53	$2.2677 * 10^{-6}$	$\mathbf{1.5409 * 10^{-6}}$	$2.2555 * 10^{-6}$
77	$2.2730 * 10^{-6}$	$1.5436 * 10^{-6}$	$2.2508 * 10^{-6}$
90	$2.2678 * 10^{-6}$	$1.5437 * 10^{-6}$	$\mathbf{2.2506 * 10^{-6}}$
92	$2.7066 * 10^{-6}$	$1.5412 * 10^{-6}$	$2.2516 * 10^{-6}$

It is possible to see that when the point corresponds to the dimension (in this case each dimension is represented by number that identifies it) the model has a lower MSE comparing to when the point number is different from the dimension number. In this way this work innovates introducing capabilities on the model in order to take in consideration the circuit' sizes.

B. *MLP Model Results*

Adopting the same strategy of the LSTM model, the train results for the MLP model with two delay lines are present in the Figure 14 and Figure 15, leading to a MSE value of $1.6254 * 10^{-7}\, V^2$ and $5.7439 * 10^{-6}\, V^2$. It is important to mention that the added noise is the same added in the LSTM predictions, having the same mean and standard deviation.

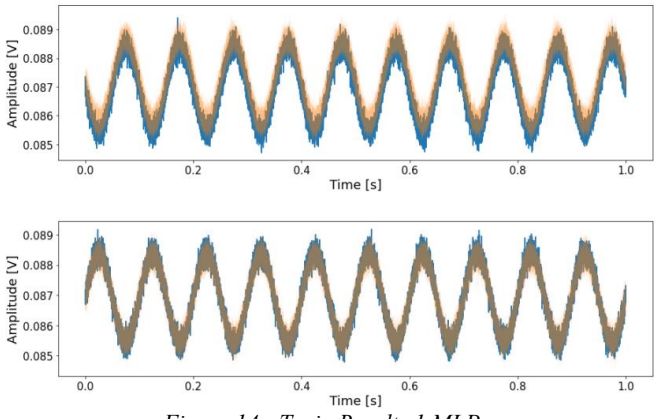

Figure 14 - Train Results 1 MLP

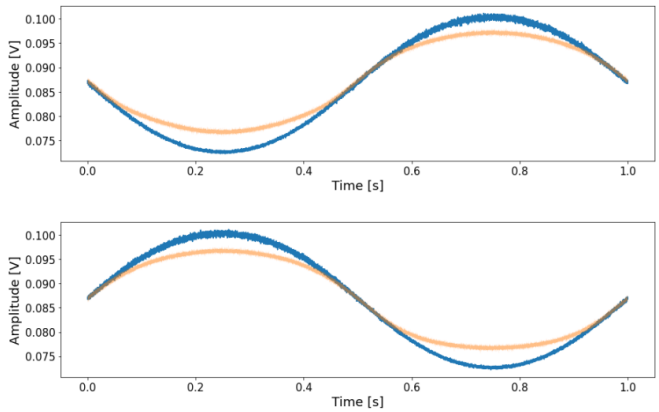

Figure 15 - Train Results 2 MLP

Knowing that the model learn it is time to see if it is capable to generalize to unseen data, so the same waves used to evaluate the LSTM model are used to evaluate the MLP model.

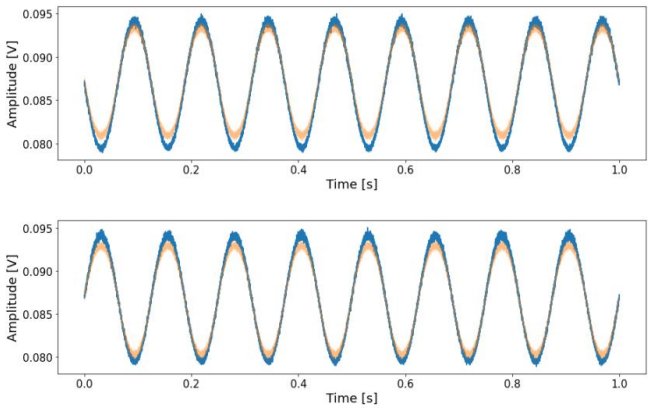

Figure 16 - Test Result 1 MLP

In Figure 16 is possible to conclude that the model can generalize for unseen data. In this test set was obtained a MSE value of $5.9297 * 10^{-7}\ V^2$.

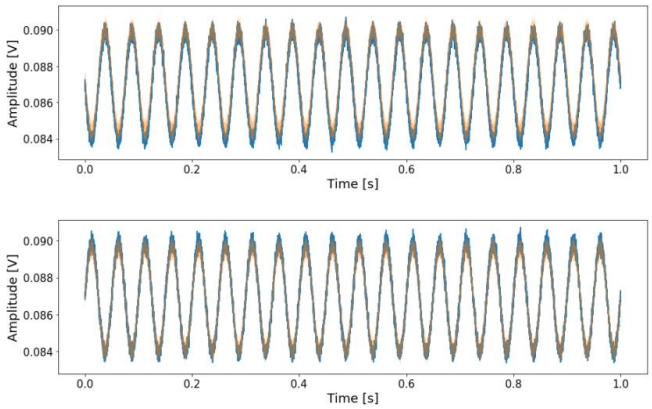

Figure 17 - Test Result 2 MLP

In Figure 17 is presented the second test set results, which, once again, proves the capabilities of the model, not only to mimic the behavior of the circuit but also to generalize to unseen data. For this dataset a MSE of $1.7319 * 10^{-7}\ V^2$ was obtained.

C. *LSTM vs. MLP Comparison*

Relatively to the sizes of the circuit' devices, similarly to what was conclude in the LSTM model results can also be conclude in the MLP model results. For the right circuit dimension, the model has a better performance, proving that the model was capable to distinguish the different sizes. In Table 5 is proven this conclusion through random points selection and the respective evaluation of the MSE.

Table 5 – Dimension Results MLP

Dim	MSE Point 0	MSE Point 53	MSE Point 90
0	$\mathbf{5.7075 * 10^{-6}}$	$1.9010 * 10^{-5}$	$5.7517 * 10^{-6}$
25	$7.6790 * 10^{-6}$	$3.3400 * 10^{-5}$	$6.5505 * 10^{-6}$
33	$7.6798 * 10^{-6}$	$3.3453 * 10^{-5}$	$6.5527 * 10^{-6}$
53	$6.1241 * 10^{-6}$	$\mathbf{1.4128 * 10^{-5}}$	$6.8016 * 10^{-6}$
77	$6.4763 * 10^{-6}$	$2.7688 * 10^{-5}$	$5.7773 * 10^{-6}$
90	$5.8932 * 10^{-6}$	$1.9226 * 10^{-5}$	$\mathbf{5.7226 * 10^{-6}}$
92	$5.8953 * 10^{-6}$	$1.9245 * 10^{-5}$	$5.7377 * 10^{-6}$

Having the MLP model working properly, it is important to make a comparison between that model and the LSTM model. Through Table 6 is possible to see that the MSE of both models are in the same order, proving the performance are similar. Relatively to the training time the LSTM took around 57 minutes and the MLP took 31 minutes to train, however this disparity of training time is explained due to the higher number of parameters the LSTM have to compute (937.452 parameters) oppositely to the MLP which has less parameters to train (186.152 parameters). Although LSTM model has a long memory and perform backpropagation through time (unfolding the network in time, making the weights being an average through time), the training phase can lead to out-of-memory errors. The MLP model with delay lines since it has a lower number of parameters do not fall in out-of-memory errors, so if the right number of delay lines is chosen it can lead to a better performance, however tuning that hyperparameter can only be done by error-trial.

Table 6 - LSTM vs. MLP

MSE	Train Set 1	Train Set 2	Test Set 1	Test Set 2
LSTM	$2.3348 * 10^{-7}$	$2.7162 * 10^{-6}$	$4.9505 * 10^{-7}$	$1.7136 * 10^{-7}$
MLP	$1.6254 * 10^{-7}$	$5.7439 * 10^{-6}$	$5.9297 * 10^{-7}$	$1.7319 * 10^{-7}$

D. *Verilog-A MLP Results*

Having the MLP model with two delay lines trained it is possible to extract the weights and bias of each fully connected layer. Using the generator script, giving the model parameters, it automatically generates the Verilog-A code. The code for the activation functions, sigmoid and ELU, is done by following the respective expressions. This process is done for all the fully connected layers and activation functions. At the end of this process all the modules are build and it is time to connected them to obtain the schematic of the circuit. The first stage is the scaling phase, the delay lines generation and the concatenation of all the waves (Figure 18)

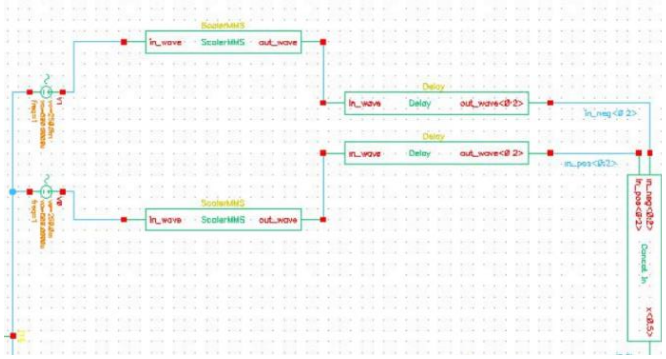

Figure 18 - First Stage Verilog-A Schematic

The second stage is the MLP model, more detailed, the fully connected layers, the sizes concatenation and the de-scaler (Figure 19).

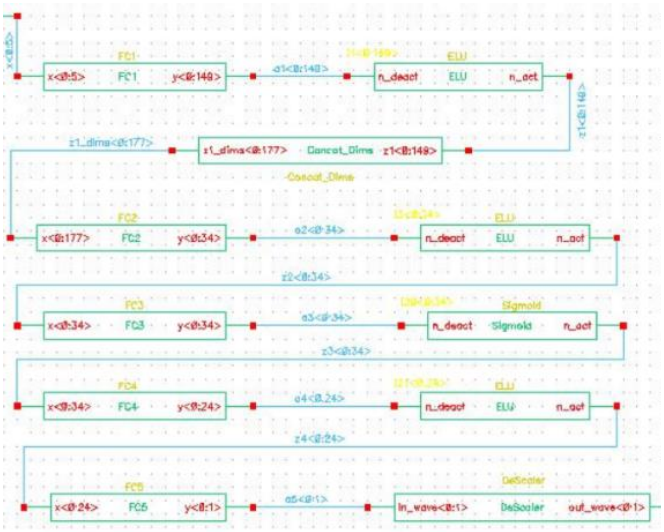

Figure 19 - Second Stage Verilog-A Schematic

The simulation is performed using the same waves used to evaluate the MLP and LSTM models. To analyse the results the blue line stands for the true label, the pink line stands for the Python prediction and the dotted yellow line stands for the spectre simulation of the Verilog-A model. It is also important to mention that the noise is added (can be seen in the schematic the module to perform that operation), however it was chosen not to be represented because it would make impossible to distinguish where is the Python prediction and the Verilog-A prediction since they will fully overlap.

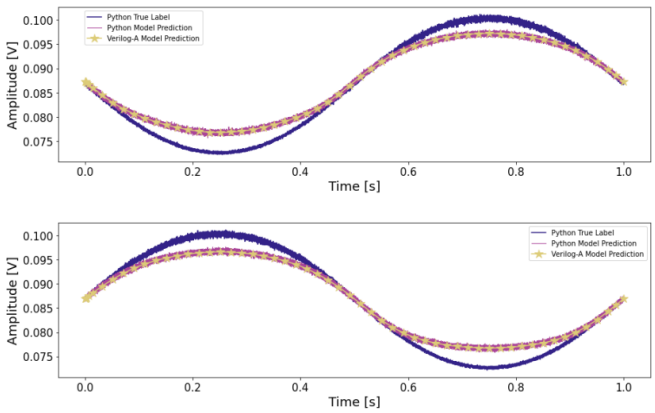

Figure 20 – Verilog-A Result 1

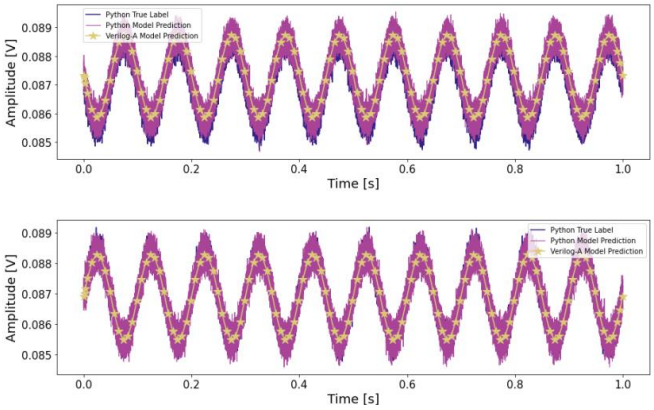

Figure 21 - Verilog-A Result 2

Figure 20 and Figure 21 present the results for the train datasets in a plot with the Python prediction, true label and the Verilog-A prediction. It is possible to conclude that the Verilog-A model is 100% accurate relative to the Python prediction, proving that the model is well implemented in this HDL.

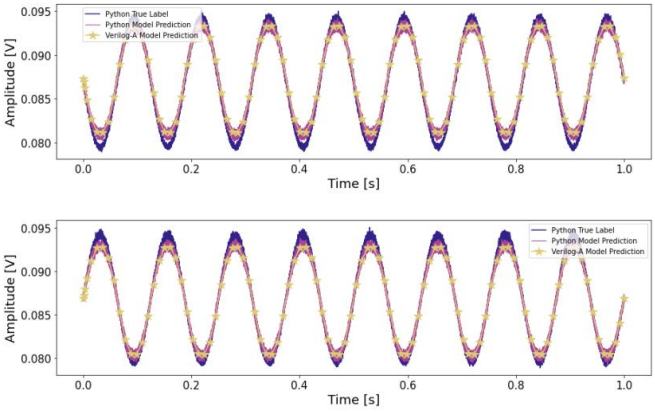

Figure 22 - Verilog-A Result 3

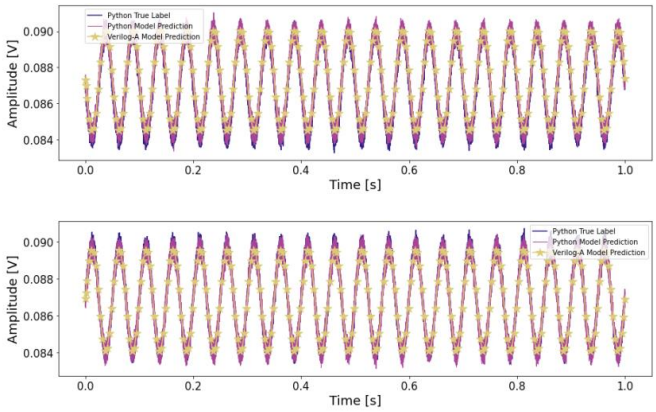

Figure 23 - Verilog-A Result 4

Figure 22 and Figure 23 present the Verilog-A results for the test sets which, once again, the Verilog-A model of the MLP can reproduce the behavior of the Python model.

Relatively to simulation time, at transistor level, it takes 2.68 seconds to generate the output and in the spectre simulation of the Verilog-A model, the time spend to obtain the output is 0.593 seconds. This proves that the model achieved the goals, being faster than the transistor level simulation, in this case, 5 times faster. To give a more insight information about the time results, at transistor level the optimization for 128 points and 500 generations takes 46 hours to obtain the output, using the Verilog-A model it takes 10 hours.

So, once more, the objectives of create a behavior model of the OTA in an HDL, capable to be faster than the transistor level, was achieved. Another achievement is the usefulness of the generator script which proves to be functional to generate Verilog-A code for artificial neural networks.

V. Conclusions and Future Work

In this paper was presented an overview of the state of art related to model the behavior of analog circuits. It was also presented two new architectures capable not only to model the circuit behavior but also to take in consideration the sizes of the circuit. A Verilog-A generator script was also made to facilitate the conversion of neural networks from Python to Verilog-A. The results of both models were also presented showing that the model can mimic the behavior for unseen data with accuracy. Relatively to the Verilog-A the results achieved prove that the MLP model implemented in Python was successfully implemented in Verilog-A.

In an overall, the work achieved the objectives proposed, bringing a new model to mimic the behavior which was the LSTM. This work allowed to explore the ANN in Verilog-A turning possible the development of a generator script which bring innovation in this research field.

As in any work, future improvements or changes can be made. In this work, pursue for a more accurate modelation of the noise can be an extension to the work, using variational auto-encoders, allowing to capture the noise behavior from the input data.

Give sense of structure to data through a Graph Neural Network (GNN) can also be a new step forward to improve this model. The GNN can provide a new type of "One Hot Encoding" allowing to have only one model capable to perform with accuracy for different typologies of circuits.

Having the ANN in Verilog-A, integrate it in the AIDA software can lead to an optimization of complex front-end circuits.

References

[1] Martins, R., Horta, N. and Lourenço, N., "Automatic Analog IC Sizing and Optimization Constrained" (1st ed.), Springer, 2017.

[2] Jordan, M. I. and Mitchell, T. M. (2015) , "Machine learning: Trends, perspectives, and prospects." Science, 349(6245), 255–260.

[3] Grabmann, M., Landrock, C. and Gläser, G. (2021), "Machine Learning in Charge: Automated Behavioral Modeling of Charge Pump Circuits". SMACD / PRIME 2021.

[4] Grabmann, M., Landrock, C. and Gläser, G. (2019), "Power to the Model: Generating Energy-Aware Mixed-Signal Models using Machine Learning". SMACD 2019, Lausanne, Switzerl.

[5] Hasani, R. M., Haerle, D., Baumgartner, C. F., Lomuscio, A. R. and Grosu, R. (2017). "Compositional neural-network modeling of complex analog circuits". International Joint Conference on Neural Networks.

[6] Yu, H., Swaminathan, M., Ji, C. and White, D. (2017). "A method for creating behavioral models of oscillators using augmented neural networks". Conference on Electrical Performance of Electronic Packaging and Systems.

[7] Yu, H., Swaminathan, M., Ji, C. and White, D. (2018). "Behavioral Modeling of Steady-State Oscillators with Buffers Using Neural Networks". Conference on Electrical Performance of Electronic Packaging and Systems.

[8] Yu, H., Swaminathan, M., Ji, C. and White, D. (2018). "A Nonlinear Behavioral Modeling Approach for Voltage-controlled Oscillators Using Augmented Neural Networks". IEEE/MTT-S International Microwave Symposium - IMS.

[9] Ceperic V., Baric A. (2004). "Modeling of analog circuits by using support vector regression machines". 11th IEEE International Conference on Electronics, Circuits and Systems.

[10] Ding M., Vemuri R. (2005). "An active learning scheme using support vector machines for analog circuit feasibility classification". 18th International Conference on VLSI Design held Jointly with 4th International Conference on Embedded Systems Design, IEEE, 2005, pp. 528–534.

[11] Chen Z., Raginsky M., Rosenbaum E. (2017). "Verilog-A Compatible Recurrent Neural Network Model for Transient Circuit Simulation". IEEE 26th Conference on Electrical Performance of Electronic Packaging and Systems.

[12] https://pytorch.org/