Debugging Low Power Analog Neural Networks for Edge Computing

Sascha Schmalhofer*, Marwin Möller*, Nikoletta Katsaouni†, Marcel H. Schulz†, Lars Hedrich*
* *Institute for Computer Science, Goethe University Frankfurt, Germany*
schmalhofer/hedrich@em.cs.uni-frankfurt.de
†*Institute for Cardiovascular Regeneration, Goethe University Frankfurt, Germany*
katsaouni/marcel.schulz@em.uni-frankfurt.de

***Abstract*—In this paper we present a method to debug and analyze large synthesized ANNs enabling a systematic comparison of the transistor netlist, behavioral model and the implementation. With that an insight into the behavior of the analog netlist is easily gained and errors during generation or badly designed cells are quickly uncovered. An overall judgement of the accuracy is also presented. We demonstrate the functionality on several examples from small ANNs to ANNs consisting of more than 10000 of cells implementing a medical application.**

I. Introduction

Analog neuronal networks (ANNs) may be a power efficient alternative to digital inference engines. We focus here on these kind of networks: [1] proposes a waferscale implementation of analog neurons in a complex net structure. [2] concentrates on resistor based matrix with about 4000 connections implementing a neural network with weights. In recent work [3] proposes circuits with 512 spiking neurons developed for inferencing tasks.

Given such a hardware-implemented big ANN, the following debugging properties and challenges arise:

- It is a complex network consisting of thousands of signals, being very slow in simulation.
- A reference model has to be extracted from the TensorFlow model and converted to the signal representation of the analog implementation which can not be done manually.
- The analog neurons itself are sensitive to load conditions and hence change the nonlinear behavior not only depending on the values handled but also on the values other connected neurons or weight cells compute.
- A complex error traversal and accumulation can occur through the network, which is hard to follow with the standard simulator output (waveforms over time).

In this paper we tackle these challenges and present a method to debug such circuits systematically enabling bug finding. See Fig. 1 for an overview of the ANN generation method and the interface to the debugging framework.

II. Previous Work

Inaccuracies for analog neural networks are analyzed in [4] on a general basis. However it has more theoretical results and incorporated these results into a TensorFlow description. A detailed view on deviation of individual cells in individual path and influence of load conditions is not handled. In [5] a more general analysis and methodologies for retraining to compensate for the loss due to inaccuracies is given.

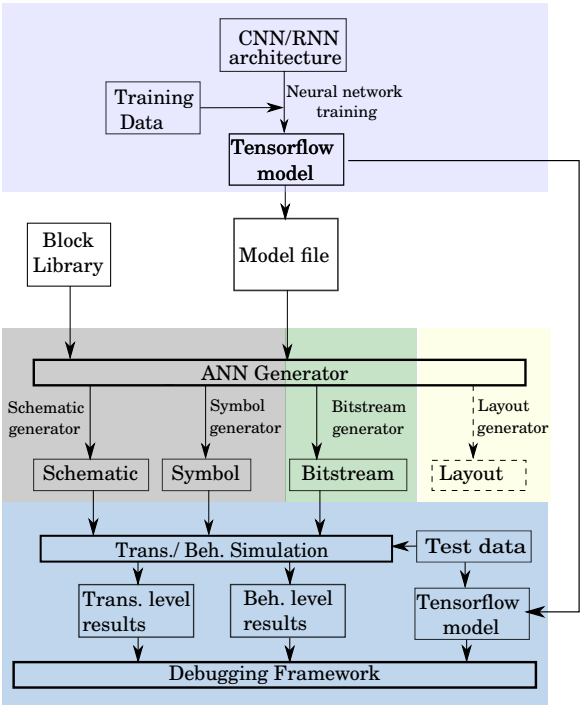

Fig. 1: Workflow of generation of analog neural net (ANN).

III. Verification Methodology

Our layer based ANN generation approach generates a top-level schematic with all layers as symbols (see Fig. 1). From that generated netlist we face the problem of verifying the whole network. The following (incomplete) list gives some impression about possible inaccuracies in the ANN:

1) Nonlinearities of transistor blocks, like current mirrors or operational transconductance amplifiers (OTAs)
2) Leakage currents influencing the wanted signal in the *pA* to *nA* range
3) Bit resolution
4) Accumulating errors in the bias-current delivery network
5) Process variations
6) Mismatch of devices
7) Parasitics: Handled by the extracted netlist

The case 3) can be handled by TensorFlow. Cases 5)-6) are not examined in this paper, but have to be cared for in future work. For all other cases a simple simulation of the overall (extracted) network at the nominal point should do the job. Unfortunately this simulation is expensive, as e.g. the ECG example with 2,190 neurons and 13,179 weights will need 4

hours for calculating only the DC solution. Hence a different strategy has to be developed.

For verification we first implemented behavioral models for nearly each cell of the whole circuit and fit and compare them to the transistor level implementation. Using these behavioral models we can speed up the transient simulation by a factor of ca. 20. However with these full behavioral simulations the real accumulating errors from the transistor netlist will not be verified. Therefore in the last step we need a transistor level simulation and an analysis where errors are resulting from thousands of signals. This is the motivation for developing a debugging framework.

IV. Debugging Framework

For the purpose of comparison between a TensorFlow-implementation of a neural network and the hardware-implementation, we observe the output of weight-backends and bias-cells and the input-values of the neuron-frontends as well as the voltages between the neuron-frontends and the weight-backends. On the TensorFlow-side of the comparison, we had to extract the output values of every single layer of the neural network.

The signals in our network switch from current domain to voltage domain and back. This is considered by using a linear mapping function from currents in the network to the TensorFlow domain ($50nA = 1(Tf)$) and a nonlinear mapping between the internal voltages of the ReLU cells and the Tanh cells to the TensorFlow domain using a piecewise linear mapping function.

In Fig. 2 an overview of the implementation of the debugging framework is given. The simulated data is pairwise preprocessed. Differences and relative measures are calculated and can be selected for displaying. To help find the source of an error/deviation, the predecessors and successors of a given cell can be highlighted. Very helpful are also statistical data generated for the overall network, e.g. deviations at output of weights binned to similar values.

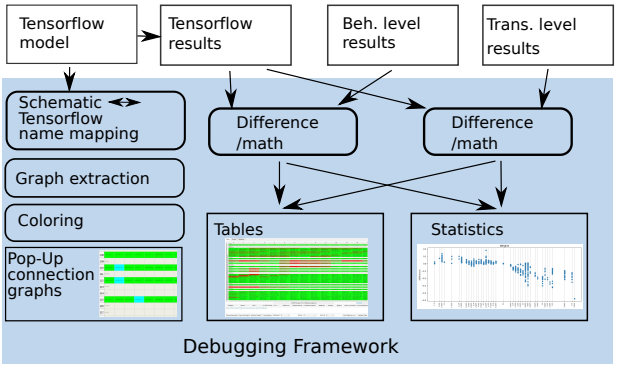

Fig. 2: Dataflow in debugging framework. The simulated data are compared pairwise with the reference data from TensorFlow.

For the transistor and behavioral level implementation a special feature to prevent wrong differences has to be incorporated: If a ReLU front end is feeded with a negative current, the voltage of that front end will drop and effectively the current into that front end will never go substantially in the negative range. As a result the feeding weights will not deliver their intended currents leading to wrong measurements. Hence these deviations are masked in the framework.

V. Results

We applied the approach on three examples (see Tab. I): a simple detector for peaks in a continuous signal, a detector for audio signals and an arrhythmia detection for electro cardiograms (ECG).

TABLE I: Examples with statistics, comparison of values at the output nodes and accuracy of implementation

Example	Peak det.	Audio det.	ECG det.
Statistics			
Inputs	11	400	896
Layers	2	6	8
Neurons/weights	19/216	611/4k	2k/13k
Deviation at output nodes			
TensorFlow ↔ behavioral			
Mean squared error	$8.8 \cdot 10^{-10}$	$2.9 \cdot 10^{-3}$	$9.6 \cdot 10^{-6}$
Maxium absolute error	$2.8 \cdot 10^{-4}$	0.154	$7.0 \cdot 10^{-3}$
TensorFlow ↔ transistor netlist			
Mean squared error	0.01	0.0007	0.0053
Maxium absolute error	0.38	0.08	0.107
Accuracy			
TensorFlow	87.2%	96.0%	62.5%
behavior	87.2%	96.0%	68%
transistor netl.	72.4%	96.0%	50%

The entries of Tab. I are comparisons between the values of the output node to the ideal TensorFlow implementation as a reference. We compare the reference with behavioral descriptions and transistor level implementations. Additionally, we could also compare the implementations at every intermediate layer with the debugging framework.

The use of the debugging framework on the ECG arrhythmia detection example than reveals some bugs, because it has the full network in focus. We fix the bugs by correcting the schematic generator.

As a consequence, the comparison of the simulation results of the netlist with behavioral models exhibits now nearly no difference to the TensorFlow implementation, which was not possible to reach without the complexity reduction due to the debugging framework.

As a result of the debugging framework, the behavioral implementation works as wanted for the audio detector and the ECG detector. For the transistor level for the ECG the debugging process shows, that errors are still accumulating. Hence further investigations, changing the circuits and/or calibration techniques are needed. The analysis for these steps are available using the framework.

References

[1] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *2008 IEEE International Joint Conference on Neural Networks*. IEEE, 2008, pp. 431–438.

[2] H. P. Graf and L. D. Jackel, "Analog electronic neural network circuits," *IEEE Circuits and Devices Magazine*, vol. 5, no. 4, pp. 44–49, 1989.

[3] B. Cramer, S. Billaudelle, S. Kanya *et al.*, "Surrogate gradients for analog neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 119, no. 4, 2022.

[4] D. Janke and D. V. Anderson, "Analyzing the effects of noise and variation on the accuracy of analog neural networks," in *63rd International Midwest Symposium on Circuits and Systems*, 2020, pp. 150–153.

[5] S. Moon, K. Shin, and D. Jeon, "Enhancing reliability of analog neural network processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1455–1459, 2019.