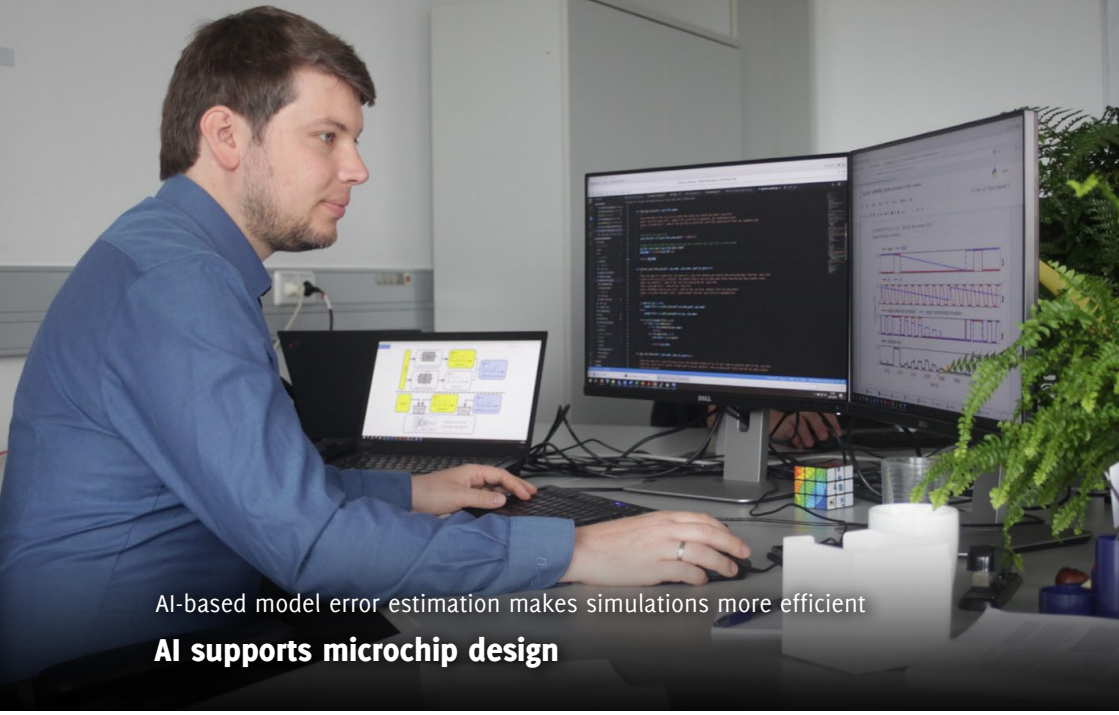

AI-based model error estimation makes simulations more efficient

AI supports microchip design

At IMMS, artificial neural networks are used to estimate model errors of behavioural models, which is essential for the simulation of integrated circuits. For system-level simulations, the approach provides reliable information about the validity of the model. Photograph: IMMS.

Motivation and overview

When designing microelectronic circuits, simulations are an essential tool for the developer, for instance, to examine the circuit design's functionality during the initial stages. However, modern circuits' complexity demands significant computational power to perform physically accurate simulations at the level of individual transistors. As a result, simplified models are frequently used, which only describe the principle behaviour of the circuit and are much easier to calculate. The drawback of these simplifications is that model errors can arise in specific situations and may result in misleading simulation outcomes.

IMMS has developed an AI-driven method within the context of the KI-EDA project. It estimates the model error of a behavioural model during simulation which may be used to employ more precise and computationally expensive simulation techniques when essential. Thus, the computational advantage in simulating behavioural models can be exploited while keeping the model error to a minimum.

www.imms.de/ki-eda

In contrast to soldering together individual components to assemble a circuit, the development of integrated circuits has a major drawback: once manufactured, there is virtually no way to modify the circuit and correct errors. While individual components on a printed circuit board can at least still be replaced with some effort, integrated circuits are manufactured monolithically on wafers in very expensive processes.

Without the possibility of a "try and error" approach, the design of integrated circuits has similarities to the construction of a bridge. Here, civil engineers must employ simulations to demonstrate safety and stability before building the structure. Similarly, it is crucial to assess the functionality of integrated circuits by employing hardware simulations before their manufacturing process.

Hardware simulations have been an essential aspect of Electronic Design Automation (EDA) since the 1970s and 1980s and have been continually evolving since then. In particular, various methods are used for simulating circuits. The classic method of using physically derived models for each transistor can almost be considered the gold standard of the industry. The results usually align well with the measured values of the actual circuit.

www.imms.de/eda

However, this approach has a drawback: It is very computationally expensive to solve the model equations numerically at the transistor level. Modern processors have several billion transistors, making it virtually impossible to run such simulations for the entire chip. But even the smaller application-specific integrated circuits (ASICs) built into cars, refrigerators, or sensors are complex enough that transistor-level simulations take several hours or even days. At the same time, many simulations are needed in the design process to test the circuit's operation under many conditions and with different inputs.

Simplification through abstraction

To solve this problem, a different approach to circuit simulation is used: The behaviour is considered only in the abstract. Simplifications can be made in the above example of simulating the structural stability of a bridge. The influence of the wind on the bridge is important, but in order to check its basic load-bearing capacity, complex flow models can be omitted for the time being. This simplifies the calculations and speeds up the design process.

In EDA, simulations of abstract circuit behaviour are essential. The computing time saved is frequently reinvested in the design process to simulate more tests, following the important motto “test early, test often”. However, the level of abstraction comes at a price: simple models of the bridge can provide solid data. But ignoring the wind completely can lead to something like the spectacular collapse of the Tacoma-Narrows Bridge in Washington State in 1940.

Knowing the model error as important information

Therefore, when simulating integrated circuits, it is essential to use behavioural models. However, one must always keep an eye on model errors and know that the model may fail under certain conditions. But how do you keep track of model errors without constantly resorting to computationally expensive transistor-level simulations?

In the KI-EDA project, IMMS has developed a method to estimate this model error during simulation. A neural network was trained to estimate the model error of a behavioral model of a circuit depending on the inputs and outputs of the model. Knowing the model error, simple behavioural models can be used whenever conditions allow and the model error is small. More sophisticated transistor-level simulations are only used when really necessary. Alternatively, the information can be used to estimate the robustness of the simulation result. This can be used, for example, in very long simulations to abort the simulation when the model error exceeds a limit. This saves unnecessary computation time for erroneous data.

www.imms.de/ki-eda

In practice, integrated circuits are divided into blocks, each of which performs a specific function. These modules, called intellectual property (IP) blocks, can be simulated individually or together as a complete chip. The basic idea of error estimation is to provide a pre-trained neural network in a separate block to an IP block for simulation. This does not affect the circuit in any way, but can read inputs and outputs of the IP block and use them as its own inputs for the error estimation. The advantage of this approach is that the error can be estimated online during the simulation, rather than having to be calculated afterwards. All common hardware simulators are able to compute these neural networks, as long as they are available in an appropriate programming language.

The method involves three steps: A) data generation, B) training, and C) setup and actual simulation. These steps are explained below using a comparator as a simple example circuit. This comparator has two main inputs and compares the voltages applied to them. Depending on which voltage is greater, the comparator switches its single output to high or low. In addition, this comparator has connections for a reset signal and the supply voltage, for a total of 5 inputs and outputs.

Data generation

A multilayer perceptron (MLP) was chosen as the learning element. This is a special type of artificial neural network that converts an input vector into an output vector and can have any number of neurons in intermediate layers. During training, the weights of the neuronal inputs and outputs of the intermediate layers are changed in a way that specific input values of the perceptron are associated with output values. Thus, this is a type of regression.

To train the MLP, both simulation data of the behavioural model of a circuit and data of a transistor-level simulation are required. If the input data for the two simulations are matched, the difference between the simulation results can be used as a measure of the model error. As in regression, the trained MLP is able to interpolate intermediate values with sufficient accuracy. Extrapolation, i.e. estimation outside the learned range of values, must be treated with caution. Therefore, it is recommended to integrate as many and different system states of the circuit as possible

into the training data. For instance, in the comparator example, it is recommended not only to apply different voltages to the main inputs, but also to vary the supply voltage and trigger the comparator reset. This allows the MLP to learn the behaviour of the circuit even in unusual situations, which is particularly useful for estimating model errors. The data set should represent the possible input space sufficiently, but does not have to cover it completely.

Training

The MLP is trained using the simulation data. This is conveniently done in a general-purpose programming language such as Python. On the one hand, hardware description languages and simulators are able to compute neural networks. However, they lack functions for iterative training. On the other hand, general-purpose programming languages provide easy-to-use tools and extensions, such as scikit-learn, that are optimised for training MLPs. However, the MLP programmed and trained in Python has to be translated into a hardware description language so that it can be integrated into a circuit. For this purpose, a tool such as Mako can be used, which can generate programme code from templates and appropriate parameters. The trained MLP is now available as a fixed block and can be integrated into a circuit and simulated. The MLP is re-trained for each circuit and cannot be combined with unfamiliar IP blocks.

Setup and simulation

The neural network is integrated into the circuit in such a way that it can read the inputs and outputs of the IP block, but does not affect the behaviour of the circuit. Thus, the MLP trained on our example comparator uses the signals of the two main inputs of the comparator, its supply voltage and reset signal, and its output as its own inputs. From these, it estimates the model error, which is not fed back into the circuit (Fig. 1). With this setup, regular tests can now be performed according to the planned development process of the integrated circuit.

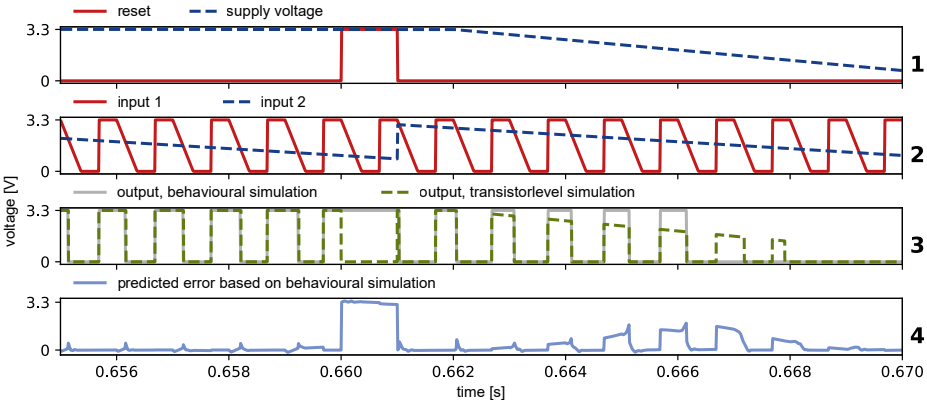

Figure 1: Example of a comparator: The device processes 4 inputs (Graph 1+2). The output was calculated once with a behavioural simulation and once with a transistor-level simulation (Graph 3). The behavioural simulation was used to estimate the behavioural model error (Graph 4). Model errors occur in this example because the behavioural model only considers the supply voltage in binary (on/off) and has a different behaviour during reset. Source: IMMS.

Model error as a measurement

The estimated model error can be used in several ways. First, it can be used as a proxy for a confidence interval of the output values of the simulated IP block. Second, it can be used as a termination criterion for the simulation. If the model error reaches pre-defined critical values, the simulation can be stopped and restarted with changed parameters. This is particularly useful for longer simulations where it would otherwise take several hours or days to obtain a result and analyse the behavioural model failure of the circuit.

To increase the effectiveness of hardware simulation, the model error can be used as a metric to dynamically apply different models to the simulation. In this case, the simulation is started with the behavioural model that is to be computed quickly. If the error of this model reaches a critical limit, the simulation is stopped and automatically restarted with the intermediate values at that point. The new simulation can be adjusted according to the model error. For example, the step size of the simulator's solvers can be changed to achieve higher accuracy, or the block can be simulated at the transistor level.

Cost-benefit analysis of AI-based error estimation

The neural networks used are very small compared to other AI methods. The error estimator of the comparator used in the example has a total of 45 neurons in two layers, no comparison to the 80-100 billion neurons of ChatGPT. Therefore, the computational cost of computing the error estimator in addition to the behavioural model is manageable. In the non-optimised comparator example, the additional computational cost of the error estimator compared to the behavioural model is 75%. However, this is offset by an additional 350% computational cost for the transistor-level simulation. Therefore, the error estimator is not free, but the behavioural model including the error estimator can still be computed significantly faster than the transistor-level simulation.

Why only estimate the error but not the whole behaviour of the circuit?

Since this method is suitable for estimating the error and thus the behaviour of the circuit, the question naturally arises as to why the neural network cannot immediately replace the entire behavioural model of the circuit in the simulation. This is due to a fundamental problem of neural networks: Traceability. In simulations, they behave like a black box, so causal relationships between input and output are difficult for humans to understand. In the bridge example used earlier, a neural network could be used to simulate the statics of the bridge. However, if the bridge collapses in the simulation, it becomes difficult to determine the exact cause and subsequently derive improvements to the design.

In addition, the comprehensibility of the behavioural models also builds trust. Unlike trained neural networks, classical equations are easier to document, understand, and communicate. Therefore, especially in certification procedures, it is advantageous to base the simulation data on conventional behavioural models and estimate only the model error by a neural model.

Practical application

The described method is able to deal with dynamic systems by considering not only current inputs and outputs, but also a time window when training the model error. However, there are special circuits that are not compatible with this method: If the behaviour of the circuit depends not only on the time history of its inputs, but also on variable intrinsic states, the MLP cannot estimate the model error correctly.

In addition, our method currently requires a separate MLP for each output of the IP block. However, this is due to the fact that the scikit-learn tool used only supports MLPs with one output. This can be solved by choosing other networks or training tools.

Since the newly developed method has proven its general suitability, it will now be tested at IMMS in the field of ASIC design in an industrial environment in order to gain further experience with it and to develop it further.

www.imms.de/eda

www.imms.de/asic

Contact person: Henning Siemen, M.Sc., henning.siemen@imms.de

More detail:

H. Siemen, M. Grabmann and G. Gläser, „Learn from error! ML-based model error estimation for design verification without false-positives," 2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Villasimius, Italy, 2022, pp. 1-4, doi: 10.1109/SMACD55068.2022.9816317.

SPONSORED BY THE

Federal Ministry of Education and Research

The KI-EDA project is funded by the Federal Ministry of Education and Research within the framework of the programme "Microelectronics for Industry 4.0 (ElektroniK I4.0)" under the consortium number es2eli4001, IMMS under the reference 16ME0010. Partners are Centitech GmbH (FRABA Group) and iC-Haus GmbH.

www.imms.de/ki-eda

KI-basierte Modellfehlerschätzung macht Simulationen effizienter

KI unterstützt die Entwicklung von Mikrochips

Am IMMS werden mit künstlichen neuronalen Netzen Modellfehler von Verhaltensmodellen geschätzt, was für die Simulation integrierter Schaltungen essenziell ist. Bei Simulationen auf Systemebene liefert der Ansatz zuverlässige Informationen über die Gültigkeit des Modells. Foto: IMMS.

Motivation und Überblick

Beim Entwurf mikroelektronischer Schaltungen sind Simulationen ein wichtiges Werkzeug für den Entwickler, um beispielsweise frühzeitig die Funktionsfähigkeit des Schaltungsentwurfs zu überprüfen. Die Komplexität moderner Schaltungen erfordert jedoch einen großen Rechenaufwand, um physikalisch genaue Simulationen auf der Ebene einzelner Transistoren durchzuführen. Daher werden oft vereinfachte Modelle verwendet, die nur das prinzipielle Verhalten der Schaltung beschreiben und wesentlich einfacher zur berechnen sind. Diese Vereinfachungen haben aber den Nachteil, dass in bestimmten Szenarien Modellfehler auftreten und Simulationsergebnisse fehlerhaft sein können.

Im Projekt KI-EDA hat das IMMS eine KI-gestützte Methode entwickelt, mit der während der Simulation der Modellfehler eines Verhaltensmodells abgeschätzt werden kann, um zeitweise und nur nach Bedarf auf genauere und somit rechenaufwändigere Simulationsmethoden zurückzugreifen. So kann der Geschwindigkeitsvorteil bei der Simulation von Verhaltensmodellen genutzt und gleichzeitig der Modellfehler minimiert werden.

www.imms.de/ki-eda

Im Gegensatz zum Zusammenlöten von Schaltungen mit einzelnen Bauteilen hat der Entwurf von integrierten Schaltungen einen Nachteil: Einmal gefertigt, hat man so gut wie keine Möglichkeit mehr, die Schaltung zu verändern und Fehler zu beheben. Während einzelne Bauteile auf einer Leiterplatte wenigstens mit etwas Mühe noch ausgetauscht werden können, werden integrierte Schaltkreise in sehr teuren Prozessen monolithisch auf Wafern gefertigt. Ohne die Möglichkeit eines „Try and Error"-Ansatzes gilt für den Entwurf integrierter Schaltungen das Gleiche wie für den Bau einer Brücke. Hier müssen Statiker bereits vor dem Bau die Stabilität und Sicherheit mithilfe von Simulationen belegen. Genauso ist es unabdingbar, beim Entwurf von integrierten Schaltungen deren Funktionsfähigkeit bereits vor der Fertigung in Hardwaresimulationen zu überprüfen.

Seit den 70ern und 80ern sind Hardwaresimulationen ein fester Bestandteil der Electronic Design Automation (EDA) und haben sich seitdem stetig weiterentwickelt. Insbesondere gibt es unterschiedliche Ansätze, Schaltungen zu simulieren. Der klassische Ansatz, für jeden einzelnen Transistor physikalisch abgeleitete Modelle zu verwenden, kann fast schon als Goldstandard der Branche bezeichnet werden. Die Ergebnisse stimmen in der Regel gut mit den Messwerten der realen Schaltung überein.

www.imms.de/eda

Dieser Ansatz hat jedoch einen Nachteil: Es ist sehr rechenaufwendig, die Gleichungen der Simulation numerisch auf Transistorlevel zu lösen. Moderne Prozessoren haben mehrere Milliarden Transistoren, sodass es so gut wie unmöglich ist, solche Simulationen für den gesamten Chip durchzuführen. Aber auch die kleineren anwendungsspezifischen integrierten Schaltungen (application-specific integrated circuits, ASICs), die zum Beispiel in Autos, Kühlschränken oder Sensoren verbaut werden, sind komplex genug, dass Simulationen auf Transistorlevel mehrere Stunden oder sogar Tage dauern. Gleichzeitig werden jedoch im Entwurfsprozess viele Simulationen benötigt, um die Funktionsweise der Schaltung unter vielen Bedingungen und mit unterschiedlichen Eingaben zu testen.

Was ist ein Modellfehler und warum ist er wichtig?

Um dieses Problem zu lösen, wird ein anderer Ansatz zur Simulation von Schaltungen verwendet: Das Verhalten wird nur abstrakt betrachtet. Im zuvor erwähnten Beispiel einer Simulation der Brückenstatik können Vereinfachungen vorgenommen

werden. Der Einfluss des Windes auf die Brücke ist zwar wichtig, doch um ihre prinzipielle Tragfähigkeit zu überprüfen, kann zunächst auf komplexe Strömungsmodelle verzichtet werden. Dies macht die Berechnungen einfacher und hilft, den Entwurfsprozess zu beschleunigen. Im Bereich der EDA sind Simulationen des abstrakten Schaltungsverhaltens essenziell. Die eingesparte Rechenzeit wird im Entwurfsprozess in der Regel dafür genutzt, mehr Tests zu simulieren, frei nach der wichtigen Devise „test early, test often“. Der Grad an Abstraktion hat aber seinen Preis: Einfache Modelle der Brücke können solide Daten liefern. Ignoriert man den Wind jedoch komplett, kann es so kommen wie mit der Tacoma-Narrows-Brücke im US-Bundesstaat Washington, die 1940 spektakulär zusammenbrach.

Bei der Simulation von integrierten Schaltungen ist es also unabdingbar, Verhaltensmodelle zu nutzen. Man muss aber immer ein Auge auf die Modellfehler haben und wissen, dass unter bestimmten Bedingungen das Modell versagen kann. Nur wie ist es möglich, die Modellfehler im Blick zu behalten, ohne ständig auf rechenaufwändige Transistorlevelsimulationen zurückzugreifen?

www.imms.de/ki-eda

Im Projekt KI-EDA hat das IMMS eine Methode entwickelt, mit der dieser Modellfehler während der Simulation abgeschätzt werden kann. Ein neuronales Netzwerk wurde darauf trainiert, den Modellfehler eines Verhaltensmodells einer Schaltung in Abhängigkeit der Eingaben und Ausgaben des Modells zu schätzen. Kennt man den Modellfehler, kann man zum Beispiel einfache Verhaltensmodelle nutzen, wann immer die Bedingungen es zulassen und der Modellfehler niedrig ist. Aufwändigere Transistorlevelsimulationen setzt man dann ein, wenn es auch wirklich nötig ist. Alternativ kann die Information auch verwendet werden, um die Belastbarkeit des Simulationsergebnisses abzuschätzen. Dies kann beispielsweise bei sehr lang dauernden Simulationen dazu genutzt werden, die Simulation abzubrechen, sobald der Modellfehler ein Limit überschreitet. Das spart unnötige Rechenzeit für fehlerbehaftete Daten.

Wie schätzt man den Modellfehler?

In der Praxis werden integrierte Schaltungen in Blöcke unterteilt, die jeweils bestimmte Funktionen übernehmen. Diese IP-Blöcke (von engl. intellectual property block) genannten Module können einzeln oder im Zusammenspiel als kompletter Chip simuliert werden. Der Grundgedanke der Fehlerschätzung besteht darin, einem IP-Block für die Simulation ein vortrainiertes neuronales Netz in einem eigenen Block

hinzuzufügen. Dieser beeinflusst die Schaltung in keiner Weise, kann die Ein- und Ausgänge des IP-Blocks mitlesen und nutzt diese als eigene Eingangsgrößen für die Schätzung. Der Vorteil dieses Ansatzes ist es, den Fehler online während der Simulation mitzuschätzen und nicht erst im Nachhinein zu berechnen. Alle gängigen Hardwaresimulatoren sind in der Lage, diese neuronalen Netze zu berechnen, solange sie in einer entsprechenden Programmiersprache vorliegen.

Wie sieht das in der Praxis aus?

Die Methode erfordert drei Arbeitsschritte: A) die Datengenerierung, B) das Training und C) das Setup und die eigentliche Simulation. Im Folgenden werden diese Schritte anhand eines Komparators als einfache Beispielschaltung erläutert. Dieser Komparator besitzt zwei Haupteingänge und vergleicht die dort anliegenden Spannungen miteinander. Je nachdem, welche Spannung größer ist, schaltet der Komparator seinen einzigen Ausgang auf high oder low. Zudem besitzt dieser Komparator Anschlüsse für ein Reset-Signal und die Betriebsspannung, in der Summe also 5 Ein- und Ausgänge.

Datengenerierung

Als lernendes Element wurde ein mehrlagiges Perzeptron (MLP) gewählt. Das ist eine spezielle Art von künstlichen neuronalen Netzen, die einen Eingabevektor in einen Ausgabevektor umwandeln und dabei eine beliebige Anzahl an Neuronen in Zwischenschichten besitzen können. Während des Trainings werden vor allem die Gewichte der neuronalen Ein- und Ausgänge der Zwischenschichten verändert, sodass spezifische Eingangswerte des Perzeptrons mit Ausgangswerten assoziiert werden. Es handelt sich somit um eine Regression.

Um das MLP zu trainieren, werden Simulationsdaten des Verhaltensmodells einer Schaltung sowie Daten einer Transistorlevelsimulation benötigt. Werden für die beiden Simulationen übereinstimmende Eingangsdaten gewählt, kann die Differenz der Simulationsergebnisse als Maß des Modellfehlers verwendet werden. Wie bei einer Regression üblich, ist das trainierte MLP in der Lage, Zwischenwerte mit genügender Genauigkeit zu interpolieren. Eine Extrapolation, also eine Abschätzung außerhalb des gelernten Wertebereichs, muss mit Vorsicht betrachtet werden. Daher empfiehlt es sich, möglichst viele und unterschiedliche Systemzustände der Schaltung in die Simulationsdaten zu integrieren. Im Beispiel des Komparators empfiehlt es sich also,

nicht nur verschiedene Spannungen an den Haupteingängen anzulegen, sondern dabei auch die Betriebsspannung zu variieren und den Reset des Komparators auszulösen. Auf diese Weise kann das MLP das Verhalten der Schaltung auch in ungewöhnlichen Situationen kennenlernen, was insbesondere für die Abschätzung von Modellfehlern von Vorteil ist. Der Datensatz sollte den möglichen Eingaberaum gut repräsentieren, muss ihn aber nicht vollkommen abdecken.

Training

Mithilfe der Simulationsdaten wird das MLP trainiert. Dies geschieht zweckmäßigerweise in einer allgemeinen Programmiersprache wie Python. Zum einen sind Hardwarebeschreibungssprachen und Simulatoren zwar in der Lage, neuronale Netzwerke zu berechnen. Für das iterative Training fehlen ihnen jedoch entsprechende Funktionen. Zum anderen stehen für allgemeine Programmiersprachen einfach zu nutzende Werkzeuge und Erweiterungen wie scikit-learn zur Verfügung, die für das Training von MLPs optimiert sind. Das in Python programmierte und trainierte MLP muss jedoch in eine Hardwarebeschreibungssprache übertragen werden, damit es in eine Schaltung integriert werden kann. Hierfür kann zum Beispiel ein Werkzeug wie Mako verwendet werden, das aus Vorlagen und entsprechenden Parametern entsprechenden Programmcode erzeugen kann. Das trainierte MLP liegt nun als fixierter Block vor und kann in eine Schaltung integriert und simuliert werden. Das MLP wird für jede Schaltung neu trainiert und kann nicht mit ihm unbekannten IP-Blöcken kombiniert werden.

Setup und Simulation

Das neuronale Netz wird so in die Schaltung integriert, dass es die Ein- und Ausgänge des IP-Blocks mitlesen kann, das Verhalten des Schaltkreises jedoch nicht beeinträchtigt. Das auf unseren Beispielkomparator trainierte MLP benutzt also die Signale der beiden Haupteingänge des Komparators, dessen Betriebsspannung und das Reset-Signal sowie dessen Ausgang als eigene Eingänge. Daraus schätzt es den Modellfehler, der nicht in die Schaltung zurückgeführt wird (Abb. 1). Mit diesem Aufbau können nun reguläre Tests entsprechend des geplanten Entwicklungsprozesses der integrierten Schaltung durchgeführt werden.

Es gibt mehrere Möglichkeiten, wie der abgeschätzte Modellfehler genutzt werden kann. Zum einen kann er als er als Ersatz für ein Konfidenzintervall der Aus-

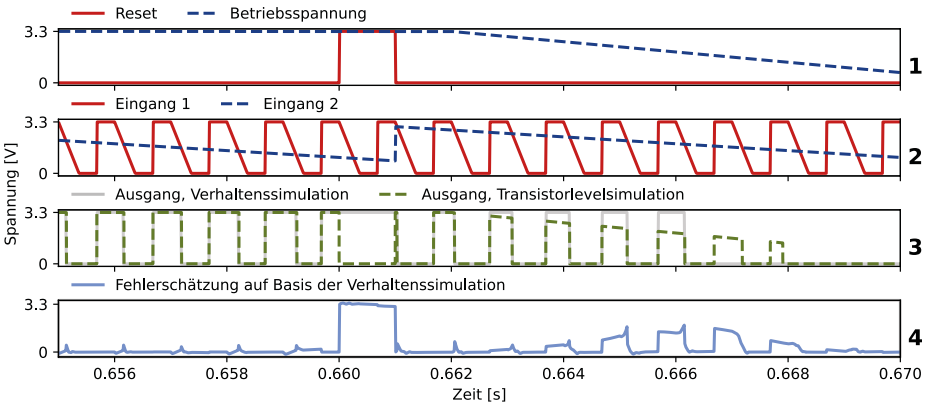

Abbildung 1: Beispiel eines Komparators: Das Schaltungselement verarbeitet 4 Eingänge (Graph 1+2). Der Ausgang wurde einmal mit einer Verhaltenssimulation und einmal mit einer Transistorlevelsimulation berechnet (Graph 3). Auf Basis der Verhaltenssimulation wurde der Fehler des Verhaltensmodells geschätzt (Graph 4). Modellfehler entstehen in diesem Beispiel, da das Verhaltensmodell die Betriebsspannung nur binär (on/off) berücksichtigt und ein anderes Verhalten während des Resets aufweist. Quelle: IMMS.

gangswerte des simulierten IP-Blocks dienen. Zum anderen kann er als Abbruchkriterium der Simulation verwendet werden. Erreicht der Modellfehler im Vorfeld definierte kritische Werte, kann die Simulation abgebrochen und mit geänderten Parametern neu gestartet werden. Dies ist insbesondere bei längeren Simulationen nützlich, bei denen ein Ergebnis sonst erst nach mehreren Stunden oder Tagen vorliegt und das Versagen des Verhaltensmodells der Schaltung analysiert werden kann. Um die Effektivität der Hardwaresimulation zu steigern, kann der Modellfehler als Messwert genutzt werden, anhand dessen dynamisch unterschiedliche Modelle für die Simulation verwendet werden. Dabei wird die Simulation mit dem schnell zu berechnenden Verhaltensmodell gestartet. Erreicht der Fehler dieses Modells ein kritisches Limit, wird die Simulation abgebrochen und automatisch mit den Zwischenwerten an dieser Stelle neu gestartet. Die neue Simulation kann entsprechend des Modellfehlers angepasst werden. So lässt sich beispielsweise die Schrittweite der Solver des Simulators verändern, um eine höhere Genauigkeit zu erreichen, oder der Block auf Transistorebene simulieren.

Aufwand und Nutzen – ein Tradeoff

Die verwendeten neuronalen Netze sind im Vergleich zu anderen KI-Verfahren sehr klein. Der im Beispiel verwendete Fehlerschätzer des Komparators hat insgesamt 45 Neuronen in zwei Schichten, kein Vergleich zu den ca. 80-100 Milliarden Neuronen von ChatGPT. Daher ist der Rechenaufwand, um zusätzlich zum Verhaltensmodell

auch noch den Fehlerschätzer zu berechnen, überschaubar. Im nichtoptimierten Beispiel des Komparators beträgt der zusätzliche Rechenaufwand des Fehlerschätzers im Vergleich zum Verhaltensmodell 75%. Dem steht jedoch auch ein zusätzlicher Rechenaufwand von 350% für die Simulation auf Transistorebene gegenüber. Die Fehlerschätzung gibt es also nicht kostenlos, aber das Verhaltensmodell samt Fehlerschätzer lässt sich immer noch deutlich schneller berechnen als die Simulation auf Transistorebene.

Warum schätzt man nur den Fehler und nicht gleich das ganze Verhalten der Schaltung?

Da diese Methode geeignet ist, den Fehler und damit auch das Verhalten der Schaltung abzuschätzen, stellt sich natürlich die Frage, warum das neuronale Netzwerk in der Simulation nicht gleich das gesamte Verhaltensmodell der Schaltung ersetzen kann. Dem steht vor allem ein grundlegendes Problem neuronaler Netze gegenüber: Nachvollziehbarkeit. Diese verhalten sich in Simulationen wie eine Blackbox, sodass für Menschen kausale Zusammenhänge zwischen Eingabe und Ausgabe nur schwer nachzuvollziehen sind. Im zuvor verwendeten Beispiel mit der Brücke könnte man die Statik der Brücke mithilfe eines neuronalen Netzwerks simulieren. Bricht die Brücke in der Simulation jedoch in sich zusammen, wird es schwer, den genauen Grund dafür zu ermitteln und anschließend Verbesserungen des Entwurfs abzuleiten. Zudem schafft die Nachvollziehbarkeit der Verhaltensmodelle auch Vertrauen. Im Gegensatz zu trainierten neuronalen Netzen lassen sich klassische Gleichungen leichter dokumentieren, verstehen und weitergeben. Daher ist es insbesondere in Zertifizierungsverfahren von Vorteil, dass Simulationsdaten auf konventionellen Verhaltensmodellen beruhen und nur der Modellfehler durch ein neuronales Modell abgeschätzt wird.

Ausblick: Wie geht es weiter?

Die Methode ist in der Lage, auch mit dynamischen Systemen umzugehen, indem beim Training des Modellfehlers nicht nur aktuelle Ein- und Ausgänge, sondern ein Zeitfenster betrachtet wird. Aber es gibt besondere Schaltungen, die mit dieser Methode nicht kompatibel sind: Ist das Verhalten der Schaltung nicht nur vom zeitlichen Verlauf seiner Eingänge, sondern auch von variablen intrinsischen Zuständen abhängig, kann das MLP den Modellfehler nicht korrekt schätzen.

Zudem wird mit unserer Methode derzeit ein eigenes MLP pro Ausgang des IP-Blocks benötigt. Dies liegt jedoch darin begründet, dass das verwendete Werkzeug scikit-learn nur MLPs mit einem Ausgang unterstützt und kann durch die Wahl anderer Netzwerke oder Trainingswerkzeuge gelöst werden.

Nachdem die neu entwickelte Methode ihre generelle Tauglichkeit belegt hat, wird sie am IMMS im Bereich des ASIC-Entwurfs im industriellen Rahmen getestet, um weitere Erfahrungen damit zu sammeln und sie weiterzuentwickeln.

Kontakt: Henning Siemen, M.Sc., henning.siemen@imms.de

Mehr Details:

H. Siemen, M. Grabmann and G. Gläser, „Learn from error! ML-based model error estimation for design verification without false-positives," 2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Villasimius, Italy, 2022, pp. 1-4, doi: 10.1109/SMACD55068.2022.9816317.

www.imms.de/eda

www.imms.de/asic

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Das Projekt KI-EDA wird vom Bundesministerium für Bildung und Forschung im Rahmen der Maßnahme „Mikroelektronik für Industrie 4.0 (ElektroniK I4.0)" unter der Verbundnummer es2eli4001 gefördert, das IMMS unter dem Kennzeichen 16ME0010. Partner sind die Centitech GmbH (FRABA Gruppe) und die iC-Haus GmbH.

www.imms.de/ki-eda