Review of Computer Engineering Research
2023 Vol. 10, No. 3, pp. 122-135
ISSN(e): 2410-9142
ISSN(p): 2412-4281
DOI: 10.18488/76.v10i3.3512

Recurrent neural network implementation of digital integrated circuits to mitigate challenges in design verification

Ming Keat Yeong[1+]
Eric Tatt Wei Ho[2]

[1,2]*Department of Electrical & Electronics Engineering, Universiti Teknologi PETRONAS, Malaysia.*
[1]*Email: ming_g03568@utp.edu.my*
[2]*Email: hotattwei@utp.edu.my*

(+ Corresponding author)

Article History
Received: 22 June 2023
Revised: 4 October 2023
Accepted: 23 October 2023
Published: 10 November 2023

Keywords
Deep learning
Design verification.
Finite state machine
Hardware validation
Model verification and validation
Neural networks.

ABSTRACT

Design verification is the dominant stage that consumes the most resources in the digital integrated circuit (IC) design process. Design verification is important because human designers imperfectly convert high-level specifications to low-level circuit implementations using standard cell logic, which is nonlinear and complex to predict and characterize. The widening process variations in shrinking process technologies while digital designs grow in scale and complexity to the extent of being impossible to fully or intuitively identify all temporal interactions of a specific design. Deep neural networks (DNN) are being progressively integrated into the sophisticated software tool chain and design process flow of digital IC as artificial intelligence can learn relationships and correlations in complex, high-dimensional, and multi-factorial problems. In this work, we propose to apply DNN to implement digital IC to minimize the complexity of digital design verification. We posit that DNN can learn to implement circuit functions directly from high-level specifications without requiring detailed specifications from the designer. Trained neural networks can be implemented on neuromorphic hardware to achieve greater power and compute efficiencies than the conventional standard cell implementation. We demonstrate that over 150 randomly generated finite state machines (FSM) can be learned effectively with Recurrent Neural Network (RNN) comprising Gated Recurrent Units (GRU) with different complexity as indicated by the number of states and inputs to the FSM. Our proposed methodology of learning RNN GRU implementations of FSM demonstrates a way forward to reduce the cost and effort of design verification, ultimately leading towards faster digital IC design cycles.

Contribution/Originality: This study proposes an innovative methodology to implement a digital integrated circuit with a deep neural network, which simplifies the design verification process. The converged deep neural network models for the same type of FSM are generalizable with 100% prediction accuracy and consistent performance, thus making the digital circuit design more flexible.

1. INTRODUCTION

Design verification is an essential step in the development of complex digital integrated circuits (ICs). The human designer may not be able to consider all possible temporal interactions between the complex logic circuitry and inputs of a circuit design, especially early in the design stage, and cannot guarantee full concordance of a design with user specifications. Functional and formal design verification are iterative processes that help the designer build confidence in the design [1, 2]. These processes include debugging to find logic errors and behaviours that

weren't meant to happen and redesigning the circuit to meet requirements. Verification contributes approximately 70% of the overall project development effort [3] and has emerged as the dominant design step to avoid costly product recalls after high-volume production. Design errors are still challenging to identify. To simplify the debug process, circuit design has evolved to include dedicated IC resources at the architectural, logic, circuit, and layout levels [4] to assist in silicon debug in design-for-debug-and-diagnosis (DFD) [5] to reduce the reliance on physical probing devices, which are costly and inefficient for debug [6]. Higher intra-die process variation in process technologies from 65nm and below [7-9] further exacerbates the design verification challenge as additional design safeguards and complex design rules must be considered to avoid unintended circuit behavior.

Functional verification is usually done with circuit simulator software to make sure the circuit works. However, this software alone can't catch all hard or subtle temporal logic errors; a human designer must list all the exact signalling conditions to find mistakes in the design. Formal verification [10] converts the circuit design to mathematical data structures such as BMDs (Binary Moment Diagrams) and utilize mathematical solvers such as Boolean SAT (Satisfiability) solvers to guarantee circuit behaviors within bounds of specified input patterns. However, formal verification suffers from exponentially increasing computation and memory demands as the complexity of the circuit scales up, i.e., in large data-path circuits like digital arithmetic blocks [11, 12]. Much of the verification effort arises because the translation from high-level specifications such as state machine diagrams and truth tables to a circuit implementation in standard cell logic is imperfect. Although sophisticated synthesis software aids in this complex and challenging process, register transfer logic (RTL) descriptions from the human designer are still necessary for increasingly detailed design input. We posit that if circuits could be designed accurately from high-level specifications, the verification effort could be reduced. The key idea of our research is to utilize deep neural networks via deep learning as a new methodology to design digital circuits. Deep neural networks are capable of modelling complex nonlinear systems efficiently. For instance, neural networks could model the circuit output behavior of a common emitter amplifier [13] with higher accuracy than conventional regression techniques and replace digital circuit blocks [14, 15]. We treat the neural network circuit as a black box that receives inputs and generates outputs, such that high-level specifications should suffice to train the deep neural network to learn the desired circuit function [16]. For example, in the standard cell IC design process, accurate physical models of standard cell logic are used to build a circuit realisation that gets more and more detailed over time. This is then thoroughly checked through a series of tests. The trained neural network should be put into a neuromorphic circuit instead of standard cell logic. This gives the circuit implementation another way to be resistant to changes in the process. In this paper, we demonstrate that a variety of random finite state machines can be implemented by Recurrent Neural Networks with only high-level specifications as input to the training.

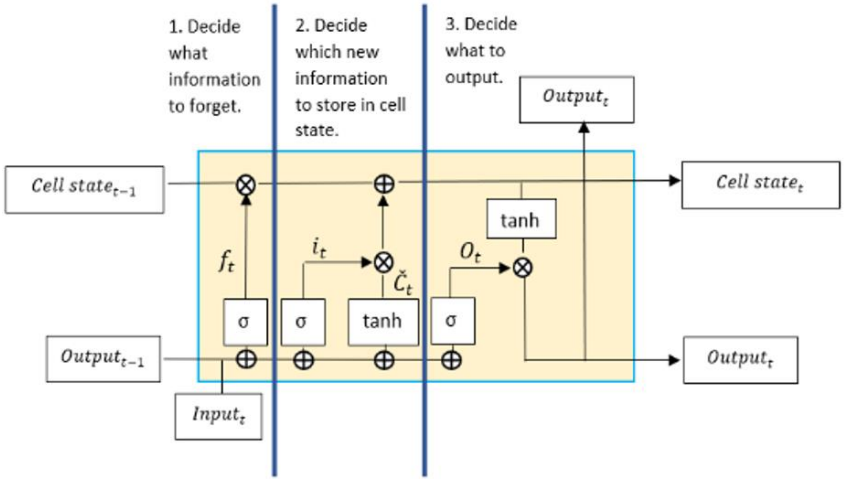

Figure 1. LSTM cell architecture and data flow.

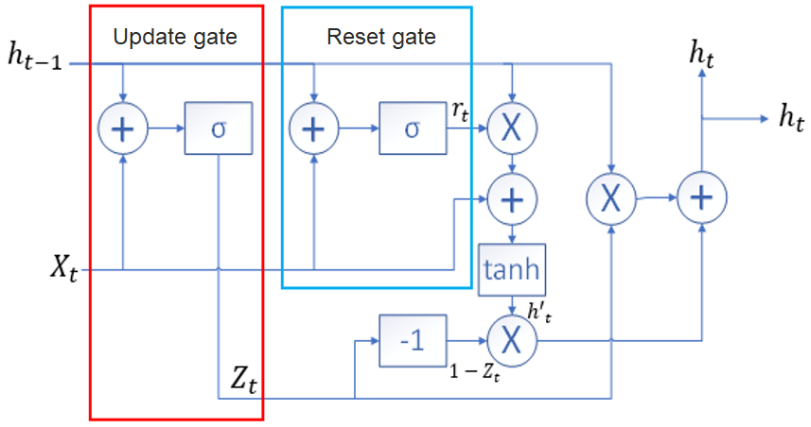

Figure 2. GRU cell architecture and data flow.

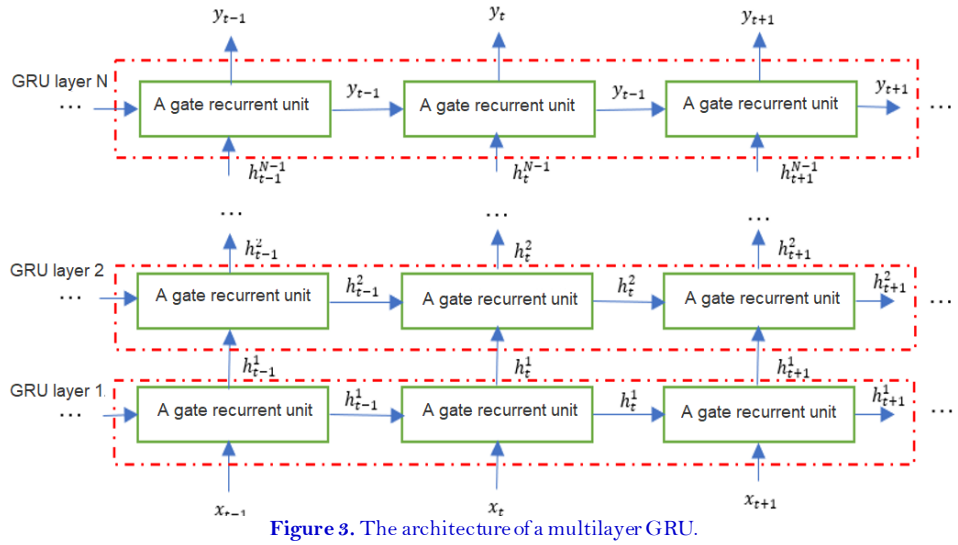

Figure 3. The architecture of a multilayer GRU.

2. RELATED WORK

The digital IC design flow is mature and standardized into five main stages: high-level synthesis (HLS), logic synthesis, placement and routing (layout), and mask synthesis. This flow is highly dependent on Electronic Design Automation (EDA), a suite of sophisticated software to automate the conversion of design specifications to actual patterns, geometries, and connections of various layers on the semiconductor substrate, but it is still highly reliant on human inputs with domain knowledge [17]. Recent advances in artificial intelligence (AI) and deep learning have been applied to improve the efficacy and accuracy of EDA tools in a variety of prediction, optimization, and generation tasks [18, 19]. AI models have been developed to improve power estimation, to develop heuristics algorithms for optimizing logic synthesis, and to improve chip routing accounting for mask optical proximity correction [17, 20, 21]. In this paper, we propose that AI models replace the circuit design process so that only high-level design specifications are sufficient to design the circuit and circumvent the extensive verification process.

2.1. Recurrent Neural Network

Neural networks that implement digital ICs require memory of past computations and the ability to redirect signals temporally. Recurrent Neural Networks (RNN) can propagate past signals into future computations but suffer from vanishing gradients that limit the memory window. Improved variants of RNN, such as Long Short-Term Memory (LSTM) [22], attempt to extend the memory window of RNN. The LSTM RNN has more layers within the feedback module compared to the standard RNN, namely the forget layer, update gate layer, and output layer. These layers transfer the cell state, or memory of previous computation, to future computations, as shown in

Figure 1 [23]. Gated Recurrent Neural Unit (GRU) is a newer generation without the internal memory cell of LSTM [24]. GRU consisted of only two gates, namely the update gate to determine the disposable information and the latest information to be added, while the reset gate identified the amount of information to forget. The architecture of GRU is simpler with fewer network parameters than LSTM, as shown in Figure 2, and therefore it trains to convergence with fewer epochs [25]. Both LSTM and GRU layers can be stacked to grow the neural network in depth and achieve better learning performance. Figure 3 shows an example of this stack for a multilayer GRU [26].

2.2. Implementing Neural Network Circuits

Neural networks can be implemented with standard cell logic, besides programmable processors and graphic processing units (GPU). Individual neurons are realized as distinct digital circuits [27] and replicated multiple times, then connected and cascaded. However, a digital implementation is not an efficient way because there are substantial challenges to implementing the large-scale connectivity at high speed and the many multiplications at each neuron [28, 29]. It is common to implement the output of a single neuron unit as a serial sum-of-product operation for all the neurons in the neural network. Some designs implement non-linear activation functions with look-up tables (LUT), range-addressable LUTs (RALUT), piecewise linear (PWL), or hybrids [30]. This type of digital implementation is also memory-intensive, as read and write are extensively used to apply neuron weight values to inputs and compute outputs [31]. For instance, AlexNet, a convolutional neural network with 8 layers, has 60 million network parameters, which require 240MB of memory for storage as 32-bit numbers [32]. These types of implementation are popularly realized in field programmable gate arrays (FPGA) [33, 34] because of the flexibility to change the neural weights for different applications. For small neural networks, FPGA implementations are faster [35] than executing software programs on microprocessors. However, FPGA implementations are restricted because gate resources are limited and the digital implementations require additional overhead logic, such as input digitization, output decoding, and clock generation [36].

Table 1. Comparison of various event-based neuromorphic chips.

Model / Features	Loihi2	Loihi	TrueNorth	Tianjic	GraAIOne (NeuronFlow)	Mythic	Brain-ScaleS	Memristor (IBM)
Signal	**Digital**	**Digital**	**Digital**	**Digital**	**Digital**	**Analog**	**Analog**	**Analog**
nm	7	14	28	28	28	40	65	50
In-memory computing	Near-memory	Near-memory	Near-memory	Near-memory	Near-memory	In-memory	Yes	Yes
On-device training	Spike-time dependent plasticity (STDP), surrogate backprop	STDP	No	No	No	-	Spike-time dependent plasticity (STDP), surrogate gradient	Yes
Signal	Real number, spikes	Spikes	Spikes	Real number, spikes	Real number, spikes	Real number	Real number, spikes	Spike
Number of cores	128	128	4096	-	-	-	-	-
Number of neurons	1 million	128,000	1 million	40,000	200,000	-	512	512
Number of synapses	120 million	128 million	256 million	10 million	-	80 million	130,000	64,000
Number of transistors	2.3 billion	2.1 billion	5.4 billion	-	-	-	-	-

Source: Ivanov, et al. [37]; Diehl, et al. [38] and Cheng, et al. [39].

To circumvent the heavy resource requirements for all-digital implementations of neural networks, researchers are exploring new types of elementary circuit building blocks for more efficient neural network implementations. Neuromorphic circuits are novel neural circuit building blocks that address the challenges of power consumption, scalability, and computation complexity in neural networks through a variety of architectural and circuit strategies. For instance, instead of LUT, the tangent hyperbolic activation function could be realized with a bipolar transistor differential pair with a PNP current mirror [36]. To avoid extensive number-to-bits encoding and achieve high-dimensional connectivity between neurons, signals are propagated as spikes [40] and integrated at the neuron. New communication elements permit highly interconnected circuits to receive and integrate spikes from a scalable number of silicon neurons (SiNs) [41, 42]. It is easy to connect multiple inputs to neuron integrators with capacitive crossbar arrays because they only use dynamic power and a lot less static power than the more common resistive crossbar array [43]. Molybdenum disulfide (MoS_2) transistors with a floating gate and two control gates can be used to make individual neurons [44] that have summation and threshold functions built in. This new neuron transistor achieves an operating frequency greater than the average firing rate of biological neurons. Neuromorphic circuits achieve lower power consumption through collocation of neurons and memory, highly parallel operation, and event-driven operation of neurons, which only consume power when processing spike inputs [45-48]. Spiking neurons activate sparsely and result in lower data bandwidth on communication buses [37].

The Neural Engineering Framework (NEF) is a general methodology for the implementation of large-scale neural networks using spiking neuron [49, 50]. NEF is widely utilized in different neuromorphic systems, such as Neurogrid, Braindrop, SpiNNaker, BrainScaleS, Darwin, Tianjic, Neuronflow, and Dynap-SEL [51]. Spiking neuron systems can be implemented either in analog or digital form. IBM's TrueNorth and Intel's Loihi [52] are examples of pure digital neuromorphic systems; NeuroGrid is a hybrid of analog and digital, while OZ is a fully analog and programmable spiking neuron system. Analog implementations of Leaky Integrate-and-Fire neurons [53] require a smaller area and are more power-efficient than the digital versions. The in-memory property of analog synapses also allows analog spiking neurons to execute much faster than digital spiking neuron, as the latter relies on near-memory approaches, for example, SRAM memory, which is placed closer to the computing core to minimize the overheads of data transaction. However, the implementation of analog spiking neurons has its challenges and is more difficult to debug as compared to digital neurons [37]. There is no long-term temporal information storage in the neuromorphic system. Table 1 summarizes the various neuromorphic systems described in the literature. To process spatiotemporal signals like electromyography (EMG) signals [46], RNN can be set up as spiking neural networks in neuromorphic hardware. Spiking RNN has also been implemented for natural language processing tasks on the TrueNorth digital neuromorphic system [38]. A conventional RNN trained with standard gradient descent and backpropagation was converted into the spiking neural network format. The RNN time delay is implemented by the digital neuromorphic hardware through a synaptic delay. This means that the feedback hidden state in the RNN travels through a predefined synaptic delay to reach the neuron input at the same time as the next input is being added. Synapses can be implemented with memristors (resistive RAM), which allow resistances to be altered by voltage/ current pulses [54].

3. METHODOLOGY

Recurrent neural network (RNN) behaves like a state machine; the current RNN output is dependent on the current input and the output from the previous time step. RNN is trained with multiple sets of state machine datasets until it can generate the correct outputs in response to the input. Randomly generated state machine datasets are used to simulate a variety of different state machines. The convergence of RNN model training with the randomly generated state machine dataset is measured with overall accuracy, which is the percentage of the total number of successful predicted next states and state outputs over the total number of states in the randomly generated state machine design. The accuracy of each state prediction is calculated with a 50% threshold, where any

output value predicted by the RNN that is higher than 0.5 will be rounded up to 1, before being compared with the expected binary output value to determine the individual accuracy at each state.

To find out the impacts of hyperparameters in the deep neural network training process and to prove the feasibility of implementing neural network architecture in circuit realization, tasks are broken down and arranged in the following sequence:

- Train a deep Gated Recurrent Unit (GRU) neural network to learn complex state machine digital circuit functionality until it achieves 100% prediction accuracy on the next state and state output based on the current state and state transition input.
- Compare the difference between LSTM and GRU models in terms of architecture and performance through the training process with randomly generated state machine datasets.
- Investigate the impacts of different neuron numbers during the training of a deep neural network for finite state machine modelling.
- To study the impact of neuron numbers on deep neural network training for the same type of state machine, 20 randomly generated state machines with the same type: 10 states with 5 inputs and 5 outputs, are used to train the deep LSTM model with the same predefined hyperparameters, except the neuron numbers.
- To study the impact of neuron numbers on deep neural network training for different sizes of state machine, 2 random state machines with different numbers of states: 16 states and 32 states, are being trained with different numbers of neurons.
- Correlate the deep neural network validation process with the real-implemented circuit verification.
- Train the converged deep GRU model with 150 randomly generated datasets for the same type of state machine (8 states, 2 transition input bits, and 2 output bits) and fine-tune the hyperparameters. This way, the model's performance can be proven consistent and used for other state machine digital circuit designs with the same type of states, inputs, and outputs.

Map the deep GRU architecture to the real digital circuit part to show that it is possible to make a real digital circuit directly from the converged deep neural network architecture.

Algorithm 1: Iterative GRU Model Training Algorithm with TensorFlow 1.15.0, Python 3.7.1

Create the GRU cell in TensorFlow
Initialize the hyperparameters
Define loss function and optimizer
Feed in the state machine training datasets to the GRU model
For each epoch in total number of epochs, do
    For each sample in the dataset, do
        Get the output from RNN
            Calculate MSE (Mean Square Error) for the gap between output and target value
    End of sample for loop
End of epoch for loop

Table 2. Common hyperparameters used by LSTM and GRU Random State machines with 8 States, 2 inputs, 2 outputs.

Hyperparameters	Values
Input bits	6
Output bits	6
Number of Neurons	100
Number of time steps	8
Batch size	1
Learning rate	0.001
Number of epochs	50

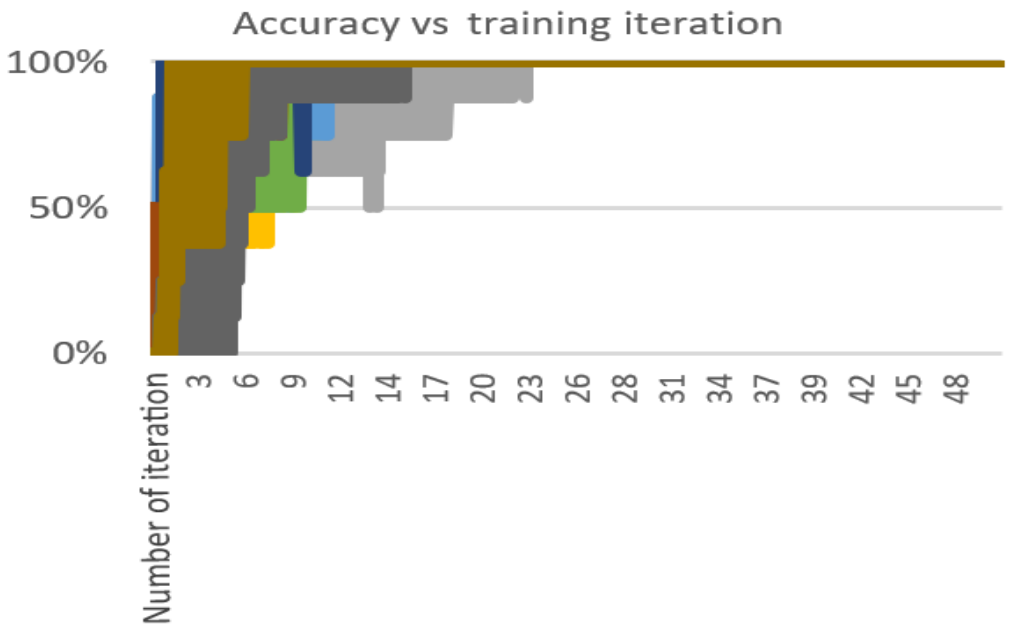

Figure 4. The accuracy of 10 distinct designs with 2-layer LSTM, which trained with hyperparameters in Table 2.

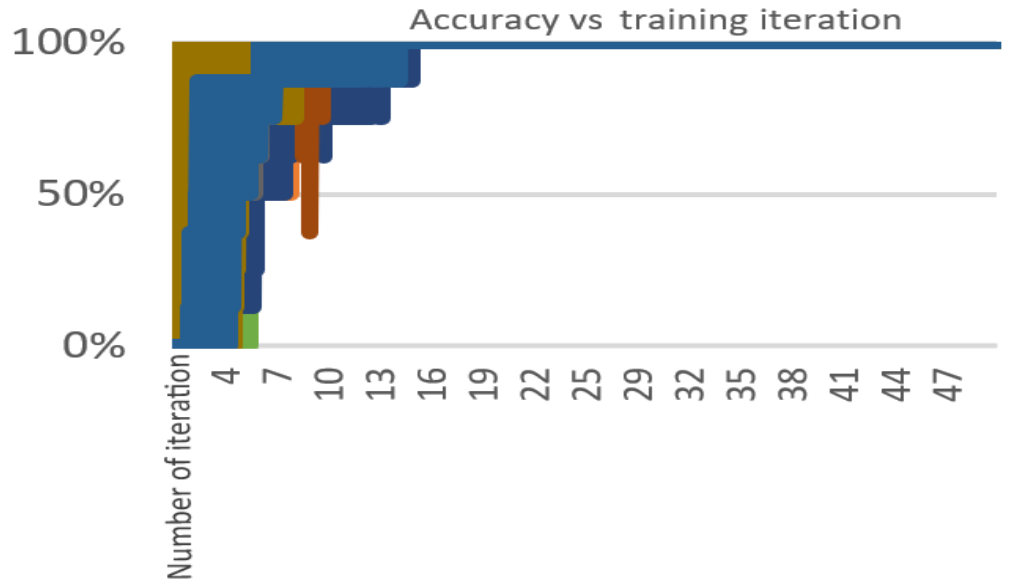

Figure 5. The accuracy of 10 distinct designs with 2-layer GRU, which trained with hyperparameters in Table 2.

Table 3. Fixed hyperparameters for the experiments to find out the relationship between neuron numbers and model training accuracy.

Hyperparameters	Values
Number of inputs	9
Number of outputs	9
Number of time steps	320
Batch size	1
Number of LSTM layer	4
Learning rate	0.001
Number of epochs	1000

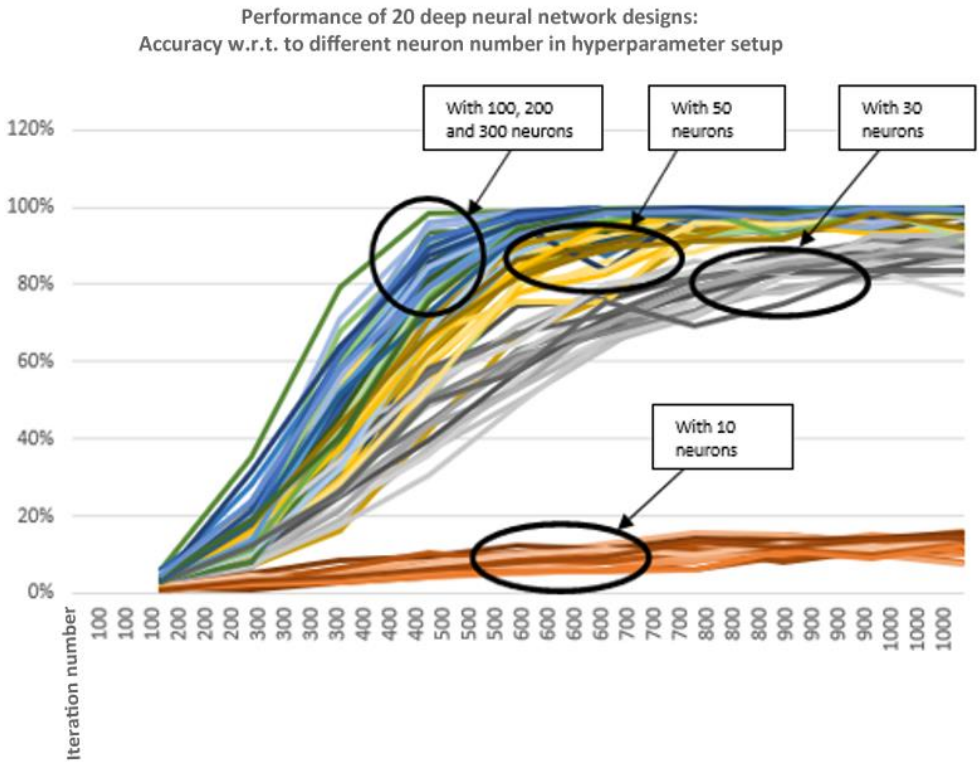

Figure 6. The trend of deep neural network convergence in response to different neuron numbers, while other hyperparameters are kept constant as shown in Table 3.

Table 4. Hyperparameters for random state machine with 16 states (a) and 32 states (b) for the experiments to study the impacts of neuron number towards deep LSTM training.

16 states	
Hyperparameters	**Values**
Number of inputs	10
Number of outputs	10
Number of time steps	512
Batch size	1
Number of LSTM layer	4
Learning rate	0.001
Number of epochs	1000

(a)

32 states	
Hyperparameters	**Values**
Number of inputs	11
Number of outputs	11
Number of time steps	1024
Batch size	1
Number of LSTM layer	4
Learning rate	0.001
Number of epochs	1000

(b)

(a) (b)

Figure 7. Comparison of impacts of neuron number towards deep LSTM training results between random state machine with 16 states (a) and 32 states (b)

4. RESULTS & DISCUSSION

4.1. LSTM and GRU RNN Successfully Implement Randomly Generated Finite State Machines

The hyper parameters in Table 2 are used to compare how well LSTM and GRU models do when they are trained with random state machine datasets. GRU model performed slightly better than LSTM model, as all the 10 random state machine designs trained with GRU can converge within 16 epochs, while LSTM took about 24 epochs, as shown in Figure 4 and 5. This is due to the simpler architecture of GRU model, which has less gates compared to LSTM and a smaller dataset, which aligns with the LSTM and GRU comparison studies in Yang, et al. [25]. In terms of realizing deep neural networks into digital circuit implementations for state machines, GRU is preferred due to its simple architecture.

4.2. Influencing Factors for Deep Neural Network Implementation of Finite State Machines

4.2.1. Number of Neurons in a Deep Neural Network vs. Size of a Finite State Machine

The relationship trend of the neuron number with the convergence rate of the neural network model is visualized in Figure 6. The speed at which deep neural network reaches convergence is proportional to the number of neuron. A deep LSTM neural network with 100 neurons and above can achieve convergence faster than one with 50 neurons and below.

Table 4 presents the hyperparameter settings that are used to train the more complex random state machines with 16 states and 32 states. The neuron numbers are being manipulated to study the impact of the neurons on the model convergence rate. The number of inputs consists of current state, previous state, and the state transition input, which are represented in binary form. For example, for the 16-state random state machine, the total number of inputs is 10, with 4 bits for current state, 4 bits for the previous state, and 2 bits for state transition inputs. While the output bits represent the output for the current state. Figure 7 summarizes the training results for the 2 different random state machines, 16 states and 32 states, with different numbers of neurons. Like the trend of a random state machine with 10 states, as shown in Figure 6, 100 neurons are sufficient to train a random state machine with 16 states, but not for the random state machine with 32 states. For random-state machines with 32 states, at least 200 neurons are required to ensure successful training of the model. This indicates that as the size of the state machine increases, more neurons are required to ensure the neural network can reach convergence.

4.2.2. Deep Neural Network Validation

Based on the hyperparameters in Table 5, the converged training result achieved during the training phase (Figure 8) can be achieved in the validation phase (Figure 9) for the 4-layer GRU model for a random state machine with 16 states. The validation approach in neural networks can validate the converged model in a much shorter time. This idea is applicable when realizing the neural network architecture into an actual digital hardware circuit, and the process of digital circuit design verification can be simplified.

Table 5. Constant hyperparameters for 4-layer GRU model validation.

Hyperparameters	Values
Number of inputs	7
Number of outputs	13
Number of time steps	16
Batch size	1
Number of GRU layer	4
Learning rate	0.001
Number of epochs	20

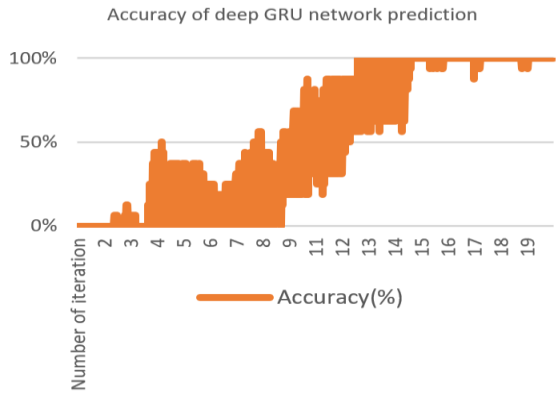

Figure 8. Training accuracy for 4-layer GRU model.

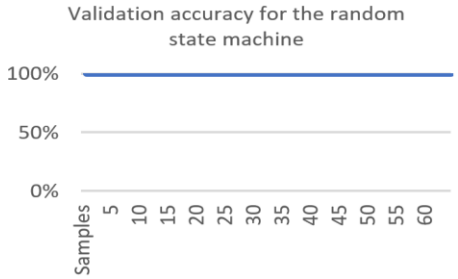

Figure 9. Validation accuracy for 4-layer GRU model.

Table 6. Fixed hyperparameters for 150 random designs with different epoch numbers for 2-layer GRU model.

Hyperparameters	Values
Number of inputs	6
Number of outputs	6
Number of neurons	100
Number of time steps	8
Batch size	1
Number of GRU layer	2
Learning rate	0.001

Table 7. Convergence rate summary for 150 random state machine designs with different epoch numbers.

Experiments overview		
Design#	**Epoch number**	**Convergence rate**
$1^{st}-50^{th}$	10	36%
$51^{st}-100^{th}$	20	98%
$101^{st}-150^{th}$	50	100%

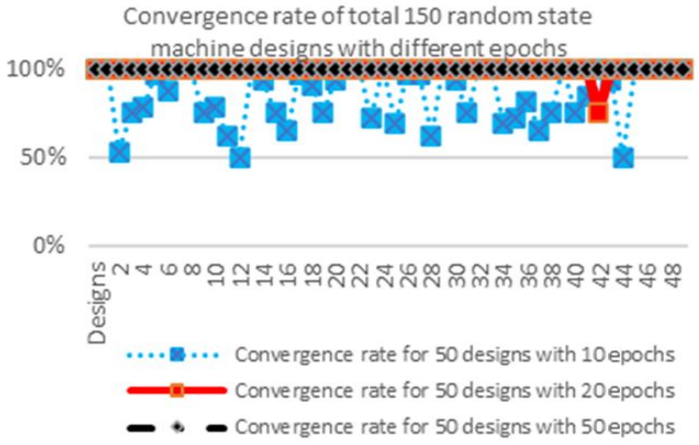

Figure 10. Visualization of convergence rate summary for 150 random state machine designs with different epochs.

4.3. Generalization of the Deep GRU for Finite State Machines

As shown in Table 6, the same hyperparameters are being used for the 150 designs training, except the epoch number, which is fine-tuned through experimentation to find the best epoch number for generalization of the deep GRU neural network for the same type of state machine designs.

Table 7 summarizes the convergence rate for the 150 random state machine trainings. The first 50 random state machine designs, which use 10 epochs, are getting a low overall convergence rate of 36%. By increasing the epoch number to 20, the overall convergence rate of the second batch of another 50 random state machine designs is significantly improved to 98%. Finally, the last round of 50 random state machines, implementing 50 epochs, can achieve a 100% overall convergence rate. Figure 10 illustrates the overall convergence rate for the 150 random state machine designs.

This generalization step is crucial to ensuring that the converged model is robust and able to adapt to different designs of the same type of digital circuit by finding reliable hyperparameters.

5. CONCLUSION

We have demonstrated evidence that Recurrent Deep Neural Networks are capable of implementing finite state machines and therefore provide a pathway to replace digital IC implemented with standard cell logic with a different, simpler methodology to be implemented on neuromorphic circuits. The RNN implementation only requires high-level design specification in the form of a finite-state machine state transition diagram and is able to implement the circuit function using a GRU RNN. We show that this methodology is generalizable by training 150 randomly generated state machines. The same set of hyperparameters can be applied to different state machines with similar complexity in the number of states and inputs. Beyond improvements in Electronic Design Automation software, we anticipate that the advances in deep neural network artificial intelligence can go further to eliminate much of the tedious work of design verification, thus simplifying the process of digital IC design, which simultaneously lowers its cost and expands access.

Funding: This study received no specific financial support.
Institutional Review Board Statement: Not applicable.
Transparency: The authors state that the manuscript is honest, truthful, and transparent, that no key aspects of the investigation have been omitted, and that any differences from the study as planned have been clarified. This study followed all writing ethics.
Competing Interests: The authors declare that they have no competing interests.
Authors' Contributions: Both authors contributed equally to the conception and design of the study. Both authors have read and agreed to the published version of the manuscript.

REFERENCES

[1] A. K. El-Sayed, S. M. Hamed, A. H. Madian, H. H. Amer, and M. B. Abdelhalim, "Structural Go/No-Go test of the TD-ADC for catastrophic faults," presented at the In 2016 International Conference on Electrical and Information Technologies, 2016.

[2] A. Dalirsani, N. Hatami, M. E. Imhof, M. Eggenberger, G. Schley, and M. Radetzki, "On covering structural defects in NoCs by functional tests," presented at the In 2014 IEEE 23rd Asian Test Symposium, 2014.

[3] H. D. Foster, "Trends in functional verification: A 2014 industry study," presented at the In 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015.

[4] T. M. Mak and S. Venkataraman, "CHAPTER 10 - design for debug and diagnosis," *In System-on-Chip Test Architectures, L.-T. Wang, C. E. Stroud, and N. A. Touba, Eds., ed* Burlington: Morgan Kaufmann, pp. 463-504, 2008.

[5] L.-T. Wang, X. Wen, and S. Wu, *CHAPTER 7 - Test synthesis," in Electronic Design Automation, L.-T. Wang, Y.-W. Chang, and K.-T. Cheng, Eds.* Boston: Morgan Kaufmann, 2009.

[6] K. Yang and K. Cheng, "Silicon debug for timing errors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 2084-2088, 2007.

[7] B. H. Calhoun, Y. Cao, X. Li, K. Mai, L. T. Pileggi, and R. A. Rutenbar, "Digital circuit design challenges and opportunities in the era of nanoscale CMOS," in *Proceedings of the IEEE*, 2008, vol. 96, pp. 343-365.

[8] Z. Xiaonan and B. Xiaoliang, "Process variability-induced timing failures— A challenge in nanometer CMOS low-power design," presented at the In 2008 IEEE International Conference on Integrated Circuit Design and Technology and Tutorial, 2008.

[9] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold, "Process and environmental variation impacts on ASIC timing," presented at the In IEEE/ACM International Conference on Computer Aided Design. ICCAD-2004, 2004.

[10] P. Bjesse, T. Leonard, and A. Mokkedem, "Finding bugs in an alpha microprocessor using satisfiability solvers.," in *In Computer Aided Verification: 13th International Conference, CAV 2001 Paris, France, July 18–22, 2001 Proceedings, Springer Berlin Heidelberg*, 2001, vol. 13, pp. 454-464.

[11] M. H. Haghbayan and B. Alizadeh, "A dynamic specification to automatically debug and correct various divider circuits," *Integration*, vol. 53, pp. 100-114, 2016. https://doi.org/10.1016/j.vlsi.2015.12.004

[12] R. Brummayer and A. Biere, "Boolector: An efficient SMT solver for bit-vectors and arrays," in In Tools and Algorithms for the Construction and Analysis of Systems: 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings, Springer Berlin Heidelberg., 2009, vol. 15, pp. 174-177.

[13] M. Dieste-Velasco, M. Diez-Mediavilla, and C. Alonso-Tristán, "Regression and ANN models for electronic circuit design," *Complexity*, vol. 2018, pp. 1-9, 2018. https://doi.org/10.1155/2018/7379512

[14] R. T. Yakkali and N. Raghava, "Neural network synchronous binary counter using hybrid algorithm training," *International Journal of Image, Graphics and Signal Processing*, vol. 9, no. 10, pp. 38-49, 2017. https://doi.org/10.5815/ijigsp.2017.10.05

[15] D. P. Sharma, "Neural network simulation of digital circuits," *International Journal of Computer Applications*, vol. 79, pp. 7-13, 2013.

[16] M. Jiao, D. Wang, and J. Qiu, "A GRU-RNN based momentum optimized algorithm for SOC estimation," *Journal of Power Sources*, vol. 459, p. 228051, 2020. https://doi.org/10.1016/j.jpowsour.2020.228051

[17] V. Hamolia and V. Melnyk, "A survey of machine learning methods and applications in electronic design automation," presented at the In 2021 11th International Conference on Advanced Computer Information Technologies (ACIT), 2021.

[18] H. Ren, "Embracing machine learning in EDA," in The Proceedings of the 2022 International Symposium on Physical Design, Virtual Event, Canada, 2022.

[19] G. Huang *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, no. 5, pp. 1-46, 2021.

[20] E. A. Gustafsson, J. A. Persson, and J. R. Ölvander, "Comparison of design automation and machine learning algorithms for creation of easily modifiable splines," in DS 101: Proceedings of NordDesign 2020, Lyngby, Denmark, 12th-14th August, 2020, pp. 1-12.

[21] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "PRIMAL: Power inference using machine learning," presented at the In 2019 56th ACM/IEEE Design Automation Conference (DAC), 2019.

[22] Z. Wu and S. King, "Investigating gated recurrent networks for speech synthesis," in In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2016, pp. 5140-5144.

[23] X. Du, H. Zhang, H. V. Nguyen, and Z. Han, "Stacked LSTM deep learning model for traffic prediction in vehicle-to-vehicle communication," presented at the In 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall), 2017.

[24] R. Al-Shabandar, A. Jaddoa, P. Liatsis, and A. J. Hussain, "A deep gated recurrent neural network for petroleum production forecasting," *Machine Learning with Applications*, vol. 3, p. 100013, 2021. https://doi.org/10.1016/j.mlwa.2020.100013

[25] S. Yang, X. Yu, and Y. Zhou, "LSTM and GRU neural network performance comparison study: Taking yelp review dataset as an example," presented at the In 2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI), 2020.

[26] Z. Yu, Z. Niu, W. Tang, and Q. Wu, "Deep learning for daily peak load forecasting–a novel gated recurrent neural network combining dynamic time warping," *IEEE Access*, vol. 7, pp. 17184-17194, 2019. https://doi.org/10.1109/access.2019.2895604

[27] M. Kolasa, R. Długosz, T. Talaśka, and W. Pedrycz, "Efficient methods of initializing neuron weights in self-organizing networks implemented in hardware," *Applied Mathematics and Computation*, vol. 319, pp. 31-47, 2018. https://doi.org/10.1016/j.amc.2017.01.043

[28] V. Gupta, K. Khare, and R. P. Singh, "FPGA design and implementation issues of artificial neural network based PID controllers," presented at the In 2009 International Conference on Advances in Recent Technologies in Communication and Computing, 2009.

[29] M. Wakamura and Y. Maeda, "FPGA implementation of Hopfield neural network via simultaneous perturbation rule," presented at the in SICE 2003 Annual Conference (IEEE Cat. No.03TH8734), 2003.

[30] C. S. Yi, W. L. Goh, Y. S. Ong, V. P. Nambiar, and A. T. Do, "Efficient implementation of activation functions for lstm accelerators," presented at the In 2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC), IEEE., 2021.

[31] Y. Maeda and M. Wakamura, "Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation," *IEEE Transactions on Neural Networks*, vol. 16, no. 6, pp. 1664-1672, 2005. https://doi.org/10.1109/tnn.2005.852237

[32] E. Nurvitadhi, D. Sheffield, S. Jaewoong, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," presented at the In 2016 International Conference on Field-Programmable Technology (FPT), 2016.

[33] N. Aamer and S. Ramachandran, "Hardware realization of neural network based controller for autonomous robot navigation," presented at the In 2017 International Conference on Computing Methodologies and Communication (ICCMC), 2017.

[34] S. O. A. Tisan, C. Gavrincea, and A. Buchman, "FPGA implementation of a self-organized map with on-chip learning," presented at the The Optimization of Electrical and Electronic Equipment, 2008. OPTIM 2008. 11th International Conference, Brasov, Romania, 2008.

[35] A. Dinu, M. N. Cirstea, and S. E. Cirstea, "Direct neural-network hardware-implementation algorithm," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 5, pp. 1845-1848, 2009. https://doi.org/10.1109/tie.2009.2033097

[36] Z. Su, B. M. Wilamowski, R. Wang, and F. F. Dai, "Adaptive integratable hardware realization of analog neural networks for nonlinear system," presented at the In 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), 2015.

[37] D. Ivanov, A. Chezhegov, M. Kiselev, A. Grunin, and D. Larionov, "Neuromorphic artificial intelligence systems," *Frontiers in Neuroscience*, vol. 16, p. 1513, 2022.

[38] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," presented at the In 2016 IEEE International Conference on Rebooting Computing, 2016.

[39] H. P. Cheng, W. Wen, C. Wu, S. Li, H. H. Li, and Y. Chen, "Understanding the design of IBM neurosynaptic system and its tradeoffs: A user perspective," presented at the In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, IEEE., 2017.

[40] C. Cerkez, I. Aybay, and U. Halici, "A digital neuron realization for the random neural network model," in *In Proceedings of International Conference on Neural Networks (ICNN'97)*, 1997, vol. 2, pp. 1000-1004.

[41] A. Hazan and E. Ezra Tsur, "Neuromorphic analog implementation of neural engineering framework-inspired spiking neuron for high-dimensional representation," *Frontiers in Neuroscience*, vol. 15, p. 627221, 2021. https://doi.org/10.3389/fnins.2021.627221

[42] G. Indiveri, B. Linares-Barranco, T. Hamilton, A. Van Schaik, R. Etienne-Cummings, and T. Delbruck, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, pp. 1-23, 2011. https://doi.org/10.3389/fnins.2011.00073

[43] J.-K. Han, J.-M. Yu, D.-W. Kim, and Y.-K. Choi, "An artificial neuron with a leaky fin-shaped field-effect transistor for a highly scalable capacitive neural network," *Advanced Intelligent Systems*, vol. 4, no. 12, p. 2200112, 2022. https://doi.org/10.1002/aisy.202200112

[44] S. Hu, Y. Liu, H. Li, T. Chen, Q. Yu, and L. Deng, "A MoS2-based coplanar neuron transistor for logic applications," *Nanotechnology*, vol. 28, no. 21, p. 214001, 2017. https://doi.org/10.1088/1361-6528/aa6b47

[45] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, pp. 10-19, 2022.

[46] Y. Ma, E. Donati, B. Chen, P. Ren, N. Zheng, and G. Indiveri, "Neuromorphic implementation of a recurrent neural network for EMG classification," presented at the In 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2020.

[47] J. Behrenbeck *et al.*, "Classification and regression of spatio-temporal signals using NeuCube and its realization on SpiNNaker neuromorphic hardware," *Journal of Neural Engineering*, vol. 16, no. 2, p. 026014, 2019. https://doi.org/10.1088/1741-2552/aafabc

[48] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018. https://doi.org/10.3389/fnins.2018.00774

[49] T. C. Stewart, "A technical overview of the neural engineering framework," Tech. Rep., Centre for Theoritical Neuroscience, University of Waterloo, 2012.

[50] K. D. Fischl, A. G. Andreou, T. C. Stewart, and K. Fair, "Implementation of the neural engineering framework on the truenorth neurosynaptic system," presented at the In 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2018.

[51] A. R. Young, M. E. Dean, J. S. Plank, and G. S. Rose, "A review of spiking neuromorphic hardware communication systems," *IEEE Access*, vol. 7, pp. 135606-135620, 2019. https://doi.org/10.1109/access.2019.2941772

[52] E. P. Frady *et al.*, "Efficient neuromorphic signal processing with resonator neurons," *Journal of Signal Processing Systems*, vol. 94, no. 10, pp. 917-927, 2022. https://doi.org/10.1007/s11265-022-01772-5

[53] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware spiking neurons design: Analog or digital?," presented at the In The 2012 International Joint Conference on Neural Networks (IJCNN), 2012.

[54] H. An, Z. Zhou, and Y. Yi, "Opportunities and challenges on nanoscale 3D neuromorphic computing system," presented at the In 2017 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI), IEEE, 2017.