
Optimizing Functionality: Deep Differential Logic Gate Networks Need Logic Initialization

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Recently, researchers have employed deep differential logic gate networks to train
2 highly efficient 1-bit neural networks capable of directly synthesizing fully 1-bit
3 neural networks into circuit chips. However, existing methods often rely on random
4 initialization connections, hindering the convergence on more complex datasets
5 and leading to lower precision. In this research, we illuminate the profound impact
6 of logic network initialization on convergence. We introduce a novel non-random
7 initialization approach that utilizes precise symbolic logic reasoning information to
8 initialize logic gate networks. Subsequently, these constructed logic gate networks
9 are converted into layer-wise logic gate networks, serving as the initialized logic
10 gate network to enhance the accuracy of differential logic gate network training.
11 Through this method, the training process gains the ability to recognize the inner
12 logic information within datasets, resulting in improved accuracy and convergence.
13 We evaluate the logic-initialized network using benchmarks from the IWLS bench-
14 mark suite, demonstrating significantly enhanced benchmark accuracy compared
15 to approaches lacking our proposed logic initialization.

16 Logic Initialization Code: <https://anonymous.4open.science/r/difflogic-init-ECE4>

17 1 Introduction

18 Recently, deep differential logic gate networks have emerged as a promising approach for 1-bit
19 lightweight DNN training, enabling the direct mapping between neural networks to circuit de-
20 signs [25, 19]. This technique can achieve a significant magnitude of improvement in neural network
21 inference speed and nearly double the enhancement in hardware logic synthesis due to its cross-
22 layer optimization. These remarkable efficiencies have garnered significant attention from AI and
23 silicon design communities, potentially revolutionizing the chip design workflow. This approach
24 allows data-driven, automated circuit chip training instead of the traditional time-consuming manual
25 design process. For instance, Deepmind utilized differential logic gate networks in the logic syn-
26 thesis competition (*i.e.* IWLS 2023[11]) and achieved state-of-the-art results compared to all other
27 techniques.

28 Although deep differential logic gate network has demonstrated significant improvements in logic
29 gate network training, they still face challenges in achieving further accuracy improvements.

30 **Challenge I: Differential Logic Inefficiency in Structural Data.** To investigate the efficiency
31 of differential logic networks on structural data, we evaluated the accuracy on the logic dataset as
32 shown in Fig. 1. The results indicate that when the differential logic network operates on logic-based
33 data, such as an 8-bit multiply operation, it underperforms compared to general neural networks.
34 Despite the smaller logic search space of 2^{32} compared to image-based datasets like MNIST, the
35 differential logic gate networks fail to match the performance levels of general neural networks.

Consequently, the current differential logic gate networks are inadequate for accurate hardware mapping of logic-based data.

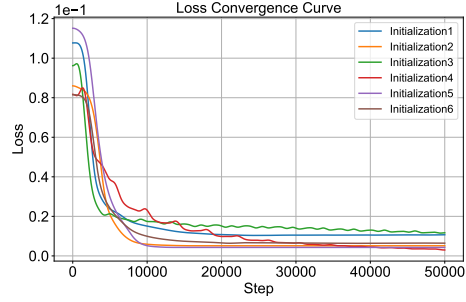
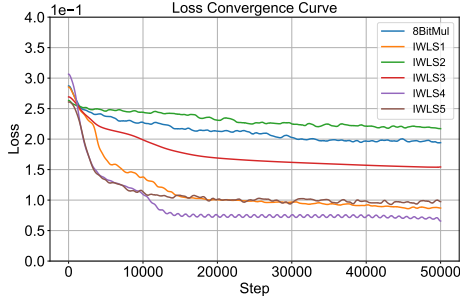


Figure 1: Loss Convergence Curves for Circuit Workloads. Figure 2: Loss Convergence Curves for Diverse Network Initializations.

Challenge II: Model Convergence Issue Attributed to Randomized Initialization. To explore whether the initialization can impact the accuracy, we tested distinct logic gate connection initializations and then trained them on the same datasets. The MSE loss function, as depicted in Fig. 2, show that different connections initialization have great impacts upon the loss function. This sensitivity indicates that randomized initialization is insufficient for effective training of differential logic gate networks.

Key Idea. Drawing inspiration from the pre-training and fine-tuning stages in current language model training, our primary innovation is to propose an initializer for the logic gate network. This initializer, used in the first stage, captures logic-based information that typical differential logic gate training cannot recognize. This approach aims to improve the accuracy and efficiency of logic gate networks.

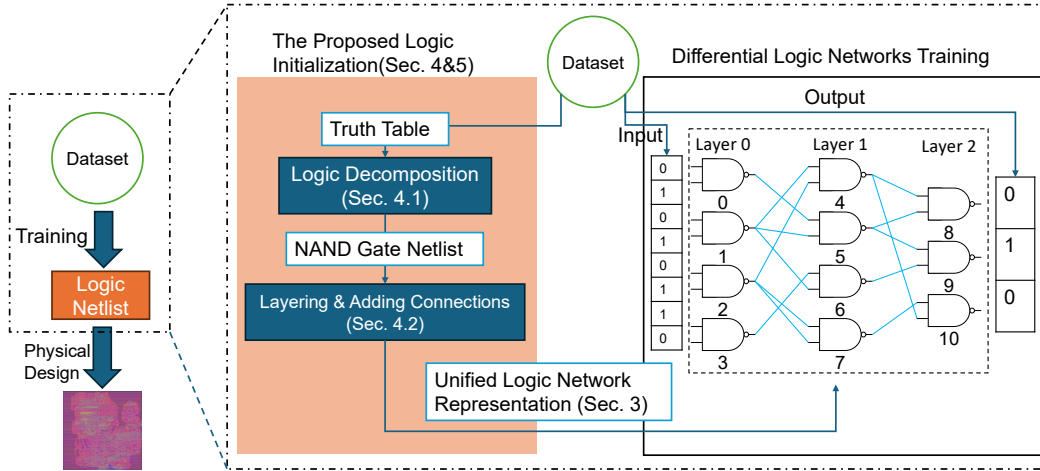


Figure 3: The Chip Design Flow with Differential Logic Gate Networks Overview.

Design. To address challenges I and II, we propose the initializer as the following three steps in Fig. 3. First, the initializer decomposes the truth table with logic decomposition, which can be reduced to the connection logic gates. Second, it converts these logic gates to a unified NAND gates-only form (a.k.a. AIG netlist[1]). Finally, the AIG network is converted into a unified matrix representation to facilitate logic gate network training. In addition, we extend this approach to sequential circuits by viewing them as recurrent differential logic networks and applying the same three stages to their recurrent form, thus enabling effective initialization for sequential circuit training.

Results. We implement the differential logic gate initialization using Python and a common logic synthesizer Yosys [26]. The evaluation compares our initialization with the differential logic network on IWLS benchmarks [20]. The results show that the average accuracy of our initialization is 82.83%,

which is better than random initialization 52.88%. This shows a promising future to directly train a dataset onto the chip circuit.

The key contributions are:

- We are the first to reveal the limitations of random connection initialization in differential logic network training, recognizing it as a major accuracy bottleneck for deep differential logic networks.
- We propose a novel initialization algorithm for deep differential logic networks that bypasses random initialization by embedding logic information directly into the network. This approach substantially enhances the accuracy of logic network training.
- We evaluate the proposed initialization framework using the IWLS benchmark. The results demonstrate that our initialization method improves accuracy from 52.88% to 82.83%, supporting the conclusion that "deep differential logic gate networks require logic initialization."

The rest of the paper is organized as follows. We first introduce the necessary background of the differential logic gate networks and logic synthesis in chip design fields in Sec. 2. Then Sec. 3 proposes a unified logic gate network representation as the intermediate representation between initialization and training stages. We introduce the primary initialization algorithm in Sec. 4 and Sec. 5. Sec. 6 evaluates the effectiveness of the initialization. Sec. 7 illustrates the limitations of this method and future work. Sec. 8 concludes this work.

2 Related Work

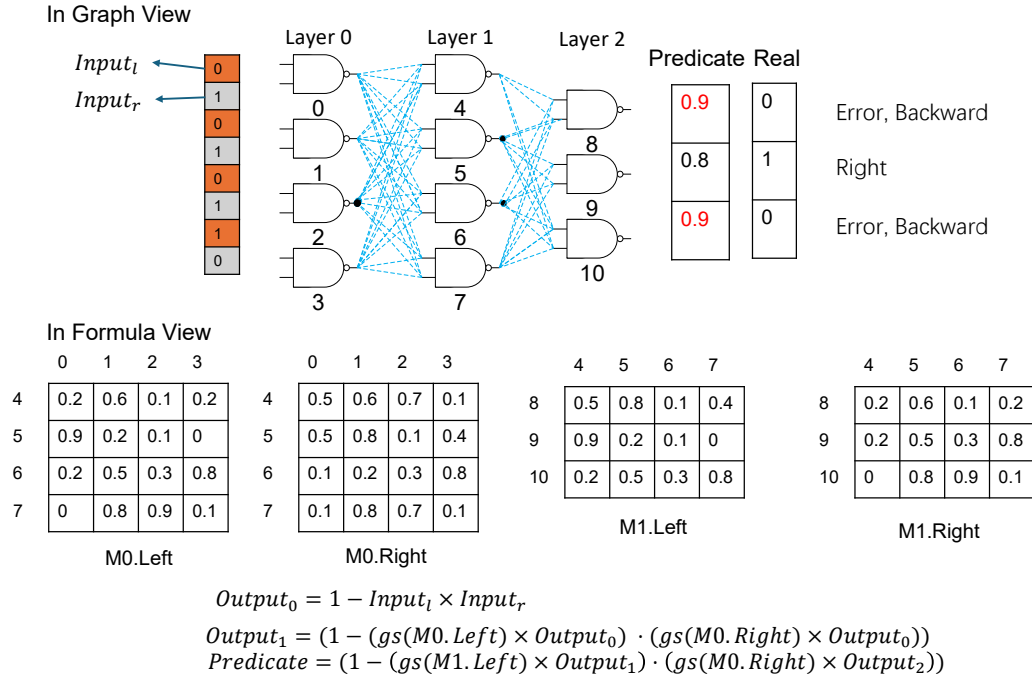


Figure 4: The representation of the differential logic networks. M0 demotes the connection matrix between layer0 and layer1. M1 demotes the connection matrix between layer 1 and layer 2. *gs* in the formula means gumbel softmax function. Left in the formula view means the left wire input. Right in formula view means the right wire input.

Differential Logic Gate Networks. Researchers have proposed a deep differential logic gate network that exclusively utilizes logic gates to avoid the costly manual chip architecture design [6]. All differential logic gate networks [18, 4, 2, 8] employ directed acyclic graphs to represent logic gate connections, similar to multilayer perceptrons (MLPs), except that the outputs are binary (0/1) during inference. These differential logic gate networks can be classified into wire stationary and

gate stationary types. The wire stationary logic gate network, as described by [18], uses random fixed connections with trainable node types, which are not compatible with hardware logic synthesis tools. In contrast, the gate stationary form, as implemented by DeepMind [4], overcomes these limitations by integrating into electronic design automation (EDA) workflows. This method has become the state-of-the-art in logic synthesis competitions, such as IWLS [4], where it converts truth tables into logic gate networks. For example, as shown in Fig. 4, the gate stationary form fixes all nodes to NAND gates with the passing function $1 - x \cdot y$. The loss of the network is defined as the mean-square error when the predicted label differs from the ground truth label. However, the differential logic gate networks still have low accuracy on the logic dataset.

Logic Synthesis. Logic synthesis is a crucial stage in the logic design process of the current EDA workflow, converting high-level representations into logic gate netlists [17, 23, 22, 5, 21]. Typical logic synthesis can be divided into two stages. The first stage translates Verilog code or truth tables into a non-optimized logic gate netlist $G = \langle V, E \rangle$ using rule-rewriting and Karnaugh maps. The second stage optimizes $G = \langle V, E \rangle$ to $G' = \langle V', E' \rangle$ through further rule-rewriting, where G' has fewer gates than G . This two-stage approach cannot globally optimize the netlist. As a result, researchers have used truth tables to approximate the synthesis of logic netlists and have proposed the IWLS benchmark[20]. As shown in Fig. 3, the datasets are first trained to logic netlist, then synthesized by the backend. However, the current approximate logic synthesis solutions can not meet the logic circuit accuracy need, which need to improve.

Neural Network Initialization. Weight initialization in neural networks has been extensively studied across various fields, leading to better convergence and improved starting points for neural networks [14, 10, 24, 3, 7]. For instance, LoRA utilizes zero initialization for partial weights [9]. Additionally, some application-specific models often use weights from general domains to pre-initialize their networks. Beyond these knowledge-based initializations, research has shown that data can significantly influence weights, prompting the development of data-dependent weight initialization methods[13]. While the aforementioned weight initialization methods have been well-explored for general neural networks, they cannot be directly applied to logic gate networks due to their non-continuous functions. Differential logic gate networks, an emerging trainable structure, have yet to be explored from an initialization perspective.

3 Workflow Overview

We depict the proposed workflow in Fig. 3. Initially, the dataset is sampled to serve as the input for the initialization stage. This initialization stage transforms the truth table into a trainable logic gate network, as elaborated in Sec. 4. Subsequently, the training stage follows the Deepmind IWLS logic gate network approach [4]. This section elucidates the bridge between the initialization and training stages, which is a unified logic gate network representation.

The unified logic gate network is a directed acyclic graph (DAG) comprising trainable connections and fixed gates. This DAG functions as expressed in Equ. 1, converting the input signal $input_i$ and the hidden state $state_i$ (also known as the current register state in cycle i) into an output signal and the next hidden state (or the next cycle register state in cycle $i + 1$). Furthermore, the network's inputs and outputs are aligned at the bit level with the hardware data.

$$\{state_i, input_i\} \rightarrow \{output_i, state_{i+1}\} \quad (1)$$

In contrast to logic representations devoid of state, the proposed unified logic incorporates a stateful representation. This establishes a connection between sequential circuits and recurrent networks, facilitating the automatic training of sequential circuits via recurrent networks. The network function is represented as a combination of layers, nodes, and edges. A layer is defined as a set of logic gate nodes with their left and right input wires, as expressed in Equ. 2.

$$Layer \triangleq Node(Edge_{left}(input), Edge_{right}(input)) \quad (2)$$

$$Output \triangleq Layer^N(Input) \quad (3)$$

Unified Node Representation. As illustrated in Fig. 4, all nodes are fixed as NAND gates due to two primary reasons. Firstly, NAND networks possess the capability to represent any logic function, thereby achieving logical completeness. Secondly, for the initialization leveraging logic recognition, logic synthesis tools employ the And-Inverter Graph (AIG) representation as the foundational form.

Unified Edge Representation. As depicted in Fig. 4, the edges connect the output port (out) of one NAND gate to the input port (in) of another NAND gate, and these connections are trainable. Specifically, for a layer with n input nodes and m output nodes, the edge can be represented as two $m \times n$ matrices, respectively corresponding to the left node input and the right node input. In these matrices, each row constitutes a one-hot vector, where the presence of 1 in the i th row and j th column denotes the connection probability. Furthermore, the edges span across layers to mitigate the potential issue of gradient diminishing.

4 Combinational Differential Logic Network Initialization

The proposed logic recognition initialization takes truth tables as inputs and outputs the corresponding unified logic gate network representation. The core concept underlying this initialization is to leverage logic decomposition techniques to convert truth tables into logic netlists, a process comprising two key steps: logic decomposition and representation conversion.

4.1 Logic Decomposition

Let us consider a truth table T with J input-output cases, wherein each case comprises an N -bit input and an M -bit output, as expressed in Equ. 4. Logic decomposition converts the truth table T into a logic gate network $G = \langle V, E \rangle$, where $V \in \text{AND, OR, NAND, XOR, NOT}$. For instance, Fig. 5 illustrates the decomposition of a four-input truth table, wherein the truth table is reduced to a basic truth table via a sequence of multiplexers. Furthermore, the input to the logic decomposition step is an incomplete truth table, implying that $|T| < 2^N$, because the dataset consists of a training set and a test set.

$$T = \{(i_0, \dots, i_N; o_0, \dots, o_M)^J\} \quad (4)$$

Given the difficulty of directly reducing large truth tables, we adopt a recursive decomposition approach, wherein the truth table is subdivided into smaller components. A key observation underpinning this method is that the truth table can be partitioned based on any input variable x . Specifically, if $x = 1$, the output corresponds to the sub-truth table tab_1 where $x = 1$; otherwise, the output is the sub-truth table tab_0 where $x = 0$. This operation can be viewed as a multiplexer, as formulated in Equ. 5. During the recursive decomposition process, the sub-truth tables eventually reduce to basic logic operators, such as AND gates, which comprise only four lines.

$$o = (!x|tab_1) \& (x|tab_0) \quad (5)$$

Fig. 5 shows a decomposition tree, where it is actually a multiplexes tree conditioning on the input variables. The decomposition output is an unnormalized logic network.

Mapping to AIG Form. To unify the generated netlist to only NAND gates(a.k.a. AIG form), the logic gate network $G = \langle V, E \rangle$ must convert to $G' = \langle V', E' \rangle$, where $V' = \{NAND\}$. The mapper rewrites the gates in V with the following rules in Tab. 1.

Table 1: Mapping rules.

Operator	New Operator
$c = a \text{ AND } b$	$t = a \text{ NAND } b; c = t \text{ NAND } t$
$c = a \text{ OR } b$	$t_0 = a \text{ NAND } a; t_1 = b \text{ NAND } b; c = t_0 \text{ NAND } t_1$
$c = a \text{ XOR } b$	$t_0 = a \text{ NAND } b; t_1 = a \text{ NAND } t_0; t_2 = b \text{ NAND } t_0; c = t_1 \text{ NAND } t_2$
$c = \text{NOT } a$	$c = a \text{ NAND } a$

4.2 Conversion to Unified Logic Gate Network Representation

The initialized logic network requires conversion to differential logic network form. This stage allocates logic gates to the locations in the differential logic network form. Consequently, this step introduces new parameters, thereby providing a sufficient number of trainable parameters for the ensuing training stage.

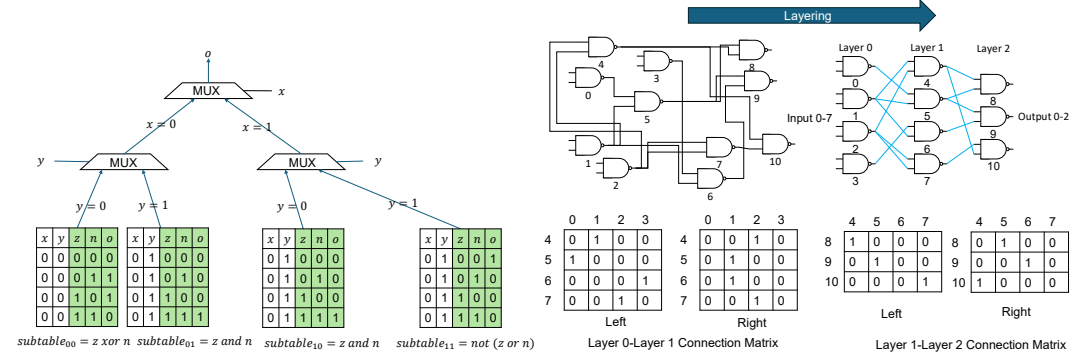


Figure 5: Decomposition Tree for four inputs and one output logic formula.

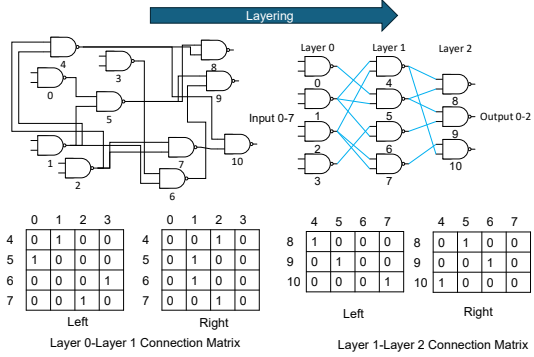


Figure 6: Layering.

Layering AIG to Unified Representation. The AIG netlist lacks the layered structure depicted in Fig. 6. However, the unified representation necessitates a layered logic gate network to facilitate the training of differential logic gate networks. This requirement is addressed through an algorithm that serves as an allocation function, mapping each NAND gate in the AIG to a distinct layer. This mapping constitutes a programming problem, with the objective of minimizing the number of layers. The dependence constraint is defined such that the execution order $gate_a < gate_b$ implies $layer_a < layer_b$, owing to the absence of cyclic connections in the AIG netlist.

We employ a greedy algorithm, outlined in Algorithm 1, to allocate each logic gate while adhering to the aforementioned objective and constraints. The algorithm commences by traversing the logic gates in their execution order, subsequently allocating each gate based on the location of its predecessors. We assume that the gate indices conform to the execution order and have been presorted according to the graph's topological order.

Algorithm 1 Gate Allocation

Input: $GateMaxIndex$, $GateInput[]$;

Output: $GateLayer[]$;

- 1: Initialize $GateLayer$ to 0;
- 2: **for all** $i = \{0, \dots, GateMaxIndex\}$ **do**
- 3: $GateLayer[i] = \max(GateLayer[GateInput[i].left], GateLayer[GateInput[i].right]) + 1$;
- 4: **end for**
- 5: **return** $GateLayer$.

Adding New Trainable Weight. To accommodate the requirement of introducing new parameters to the pre-initialized logic network, we employ a merge strategy at each layer, as illustrated in Figure 8. Specifically, for an initialized logic network layer j consisting of n NAND gates, we enhance this layer by incorporating m new gates. Consequently, the layer now comprises a total of $n + m$ nodes, effectively extending the training space.

5 Sequential Differential Logic Network Initialization

The aforementioned initialization is applicable only to logic networks comprising combinational logic. However, sequential logic circuits are crucial in real-world digital circuit environments and

189 have been largely overlooked in previous research. To bridge this gap, this section first introduces
 190 a recurrent logic network to enable the training of sequential logic circuits. Subsequently, We then
 191 extend the proposed logic network initialization to the proposed recurrent logic network, thereby
 192 enhancing the accuracy of synthesizing sequential logic networks.

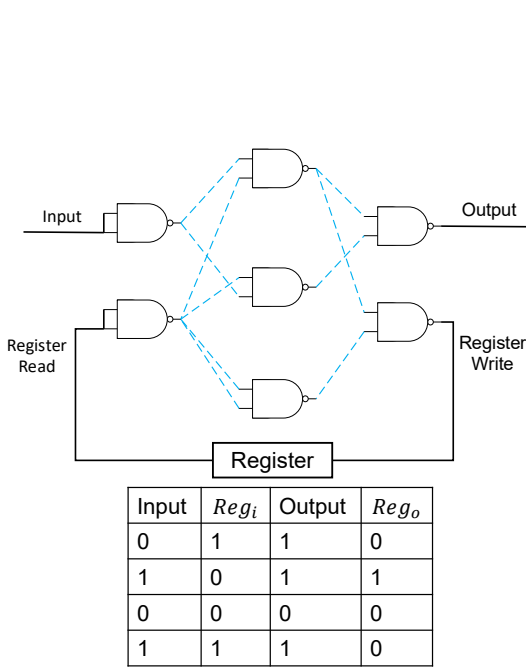


Figure 7: Recurrent Networks Representation.

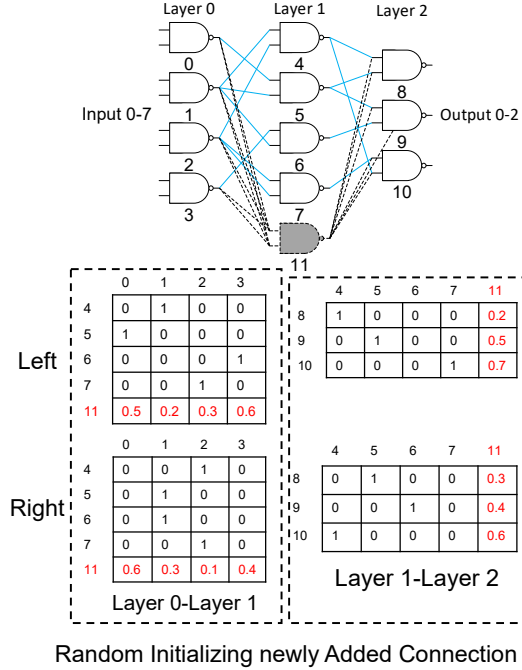


Figure 8: Add Weight.

193 5.1 Recurrent Logic Network

194 The sequential circuit can be better represented by recurrent logic networks. As shown in Fig. 7,
 195 the inputs of the sequential circuit are $iwire_i, reg_i$, where these two inputs are in i th cycle. The
 196 outputs of the sequential circuit are $owire_i, reg_{i+1}$, where these two outputs are the output wire
 197 and register value at the $i + 1$ th cycle. Specifically, the outputs can be obtained from inputs by
 198 $output, reg_N = Network^N(input, reg)$, which means the network infers N times can generate the
 199 output.

200 5.2 Sequential Logic Network Initialization

201 As depicted in Figure 7, the register values at each cycle and the input wire signals at each cycle are
 202 specified by the datasets. Consequently, the initialization can still utilize the truth table as input. Thus
 203 the initialization takes the $(wire, reg)$ as the inputs and outputs, which uses the truth table formatting
 204 as Equ. 6. The remaining steps are identical to those in the initialization of combinational differential
 205 logic networks.

$$\{iwire_i, reg_i\} \rightarrow \{owire_i, reg_{i+1}\} \quad (6)$$

206 6 Evaluations

207 6.1 Experiment Setup

208 In the initialization stage, we use OpenRoad-Yosys [15, 26] to synthesize the logic gate network to
 209 AIG netlist and do logic decomposition. In the training stage, we use the basic logic network as
 210 the same as Deepmind used in IWLS competition [11], where the logic layers are connected with

cross-layer connections to prevent gradient diminishing. The nodes are all NAND gates and the nodes' input wires are trainable, which uses gumbel_softmax [12, 27] to select the input wire. We use NVIDIA A100 and Pytorch [16] to train the logic network. The final accuracy is the results after 50k epochs, where the loss is stable. We initialize the $\tau=0.9$ in the Gumbel-softmax function.

6.2 Performance Results

The experiment shows that for most cases, our initialization can outperform than the results with no initialization. Specifically, as Tab. 2 shows, the average accuracy of initialization is 82.83% compared with 52.88% of random-initialized. We speculate that this notable increase is because the logic information has been contained in pre-initialized weights, and the training stage only needs to deal with the additional IO examples that are not initialized. To further analyze the experiment results, we illustrate two key observations in the experiment.

The initialization stage provides enough information to the logic gate network compared with the random-initialization counterpart, leading to overall loss decreasing. Specifically, from the converging perspective, the loss decreasing can better converge to a stable point. Therefore, these results suggest a performance increase after logic initialization.

Table 2: The comparison between the initialized version(*i.e.* w/ initialization) and the random initialized version (*i.e.* w/o initialization).

	rand1	rand2	s-box	Majority	sorters	Espresso	Arithmetic	LogicNets
w/o initialization	67.12%	57.44%	66.42%	38.48%	50.34%	51.61%	46.85%	44.79%
w/ initialization	81.21%	86.55%	80.30%	83.57%	83.82%	83.00%	82.69%	81.53%

6.3 Sensitivity Study

There are several factors that contribute to our experiment results. For example, the sampling data number and strategy before the initialization stage, the logic gate numbers in the differential logic gate network(*i.e.* network scale), and the epoch number when training the initialized network. This section explores how these factors take affect on our final results.

Sampling Data. The initialized algorithm needs a truth table, where the number of data in this truth table is sampled from the complete input-output examples. The sampling number relates to the initialization truth table size. The results of the sampling number change are shown in Tab. 3. Specifically, when the average sampling number increases from 40% to 90% in s-box case, the total accuracy changes from 51.05% to 90.15%. We speculate that the sampling number increase will result in an accuracy change on the test set, which means the data number improvement will result in an accuracy improvement.

Table 3: The sample data ratio change in the truth table.

	rand1	rand2	s-box	Majority	sorters	Espresso	Arithmetic	LogicNets
w/ initialization(40%)	57.04%	57.52%	51.05%	61.01%	57.50%	57.53%	57.44%	54.40%
w/ initialization(60%)	72.26%	70.60%	69.59%	67.60%	73.51%	69.59%	73.16%	69.82%
w/ initialization(80%)	81.21%	86.55%	80.30%	83.57%	83.82%	83.00%	82.69%	81.53%
w/ initialization(90%)	90.35%	90.50%	90.15%	90.68%	91.23%	90.83%	90.86%	90.95%

We classify the possible sampling data solutions into randomized and serial. (1) In serial order, we do not randomize the data, which means the data are trained with the increment order. For example, the input is 000 and is trained in front of the input is 001. (2) In random order, we select input in random order. We compared these two orders. As shown in Tab. 4, the results show that random order can have a uniform loss decreasing compared with serial order, thus we recommend serial order.

Table 4: The sample order has effects on the accuracy.

	rand1	rand2	s-box	Majority	sorters	Espresso	Arithmetic	LogicNets
Random initialization	81.21%	86.55%	80.30%	83.57%	83.82%	83.00%	82.69%	81.53%
Sequential initialization	83.74%	88.37%	80.86%	83.73%	86.12%	85.12%	86.96%	83.63%

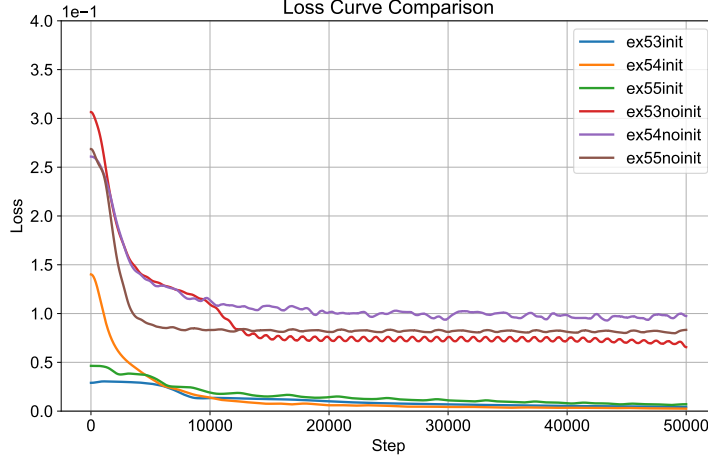


Figure 9: Comparison of Loss Curves: with initialization vs. without initialization.

Network Scale. Beyond the weight initialization, the network gate number poses an important role in the training, which can influence the network depth and width. We explore the accuracy change with the gate number as Tab. 5 shows. The results show that with the network scale increase, the accuracy decreases. Specifically, the depth changes from 10 to 20 can result in an accuracy change from 87.98% to 85.77%. The results show that the network scale can have an influence on the accuracy.

Table 5: The accuracy change with the network scale change.

Layers	0-10	10-20	20-30	30-40
Case numbers	4	37	53	6
Accuracy	87.98%	85.77%	83.04%	83.55%

Epoch Number. To show how the training epoch number takes affects on the accuracy, we do experiments on several IWLS cases. Fig. 9 shows loss curves in several cases. The results show that with the epoch number increase, the loss can decrease as no initialization. Therefore, we speculate that our initialization method preserves efficient convergence for most cases, which do not have much influence on the training stage.

7 Current Limitations and Opportunities

Large-Scale Hardware Design. As processor scales continue to increase, the accuracy of gate networks tends to diminish. This limitation arises from the inherent accuracy challenges faced by differential logic when considering an excessive number of gates. The proposed initialization cannot effectively handle excessively large networks, as the truth table size grows exponentially, leading to memory constraints. Consequently, there remains substantial room for improvement in fully automated chip design.

8 Conclusion

In this work, we reveal that randomly initialized connections are sensitive to logic knowledge, which may lead to suboptimal convergence. To address this issue, this paper proposes an initialization method for differential logic gate networks that leverages logic decomposition to recognize logic knowledge from the datasets. Additionally, we convert the initialization to the training stage via a unified representation, which can be applied to sequential logic networks. This paper evaluates the logic initialization on the IWLS benchmark suite. The results demonstrate that this initialization can enhance accuracy from 52.88% to 82.83% on average.

References

- [1] Armin Biere. The aiger and-inverter graph (aig) format version 20071012. 2007.
- [2] Tobias Brudermueller, Dennis L Shung, Adrian J Stanley, Johannes Stegmaier, and Smita Krishnaswamy. Making logic learnable with neural networks. *arXiv preprint arXiv:2002.03847*, 2020.
- [3] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *Advances in neural information processing systems*, 29, 2016.
- [4] Deepmind. Designing better computer chips, 2023. <https://deepmind.google/impact/optimizing-computer-systems-with-more-generalized-ai-tools/>, Last accessed on 2024-5-20.
- [5] David Gregory, Karen Bartlett, Aart deGeus, and Gary Hachtel. Socrates: A system for automatically synthesizing and optimizing combinational logic. In *Papers on Twenty-five years of electronic design automation*, pages 580–586. 1988.
- [6] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [7] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. *Advances in neural information processing systems*, 31, 2018.
- [8] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. Drills: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 581–586. IEEE, 2020.
- [9] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*, 2021.
- [10] Kelli D Humbird, J Luc Peterson, and Ryan G McClarren. Deep neural network initialization with decision trees. *IEEE transactions on neural networks and learning systems*, 30(5):1286–1295, 2018.
- [11] IWLS. Iwls programming contest series machine learning + logic synthesis, 2023. <https://www.iwls.org/contest/?ref=maginitive.com>, Last accessed on 2024-5-20.
- [12] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [13] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. *ICLR*, 2016.
- [14] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.
- [15] Openroad. Openroad, 2023. <https://theopenroadproject.org/><https://theopenroadproject.org/>, Last accessed on 2024-5-20.
- [16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [17] Zehua Pei, Fangzhou Liu, Zhuolun He, Guojin Chen, Haisheng Zheng, Keren Zhu, and Bei Yu. Alphasyn: Logic synthesis optimization with efficient monte carlo tree search. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

- 315 [18] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic
316 gate networks. *Advances in Neural Information Processing Systems*, 35:2006–2018, 2022.
- 317 [19] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary
318 neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.
- 319 [20] Shubham Rai, Walter Lau Neto, Yukio Miyasaka, Xinpei Zhang, Mingfei Yu, Qingyang Yi,
320 Masahiro Fujita, Guilherme B. Manske, Matheus F. Pontes, Leomar S. da Rosa, Marilton S.
321 de Aguiar, Paulo F. Butzen, Po-Chun Chien, Yu-Shan Huang, Hoa-Ren Wang, Jie-Hong R. Jiang,
322 Jiaqi Gu, Zheng Zhao, Zixuan Jiang, David Z. Pan, Brunno A. de Abreu, Isac de Souza Campos,
323 Augusto Berndt, Cristina Meinhardt, Jonata T. Carvalho, Mateus Grellert, Sergio Bampi, Aditya
324 Lohana, Akash Kumar, Wei Zeng, Azadeh Davoodi, Rasit O. Topaloglu, Yuan Zhou, Jordan
325 Dotzel, Yichi Zhang, Hanyu Wang, Zhiru Zhang, Valerio Tenace, Pierre-Emmanuel Gaillardon,
326 Alan Mishchenko, and Satrajit Chatterjee. Logic synthesis meets machine learning: Trading
327 exactness for generalization. In *2021 Design, Automation Test in Europe Conference Exhibition*
328 *(DATE)*, pages 1026–1031, 2021.
- 329 [21] Tsutomu Sasao. *Logic synthesis and optimization*, volume 2. Springer, 1993.
- 330 [22] Zdenek Vasicek and Lukas Sekanina. Evolutionary approach to approximate digital circuits
331 design. *IEEE Transactions on Evolutionary Computation*, 19(3):432–444, 2014.
- 332 [23] Swagath Venkataramani, Amit Sabne, Vivek Kozhikkottu, Kaushik Roy, and Anand Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Proceedings of the 49th Annual Design Automation Conference*, pages 796–801, 2012.
- 333 [24] Huan Wang, Can Qin, Yue Bai, Yulun Zhang, and Yun Fu. Recent advances on neural network
336 pruning at initialization. *arXiv preprint arXiv:2103.06460*, 2021.
- 337 [25] Xuan Wang, Zheyu Yan, Chang Meng, Yiyu Shi, and Weikang Qian. Dasals: Differentiable
338 architecture search-driven approximate logic synthesis. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.
- 339 [26] Claire Wolf. Yosys open synthesis suite. <https://yosyshq.net/yosys/>.
- 341 [27] Valentina Zantedeschi, Matt Kusner, and Vlad Niculae. Learning binary decision trees by
342 argmin differentiation. In *International Conference on Machine Learning*, pages 12298–12309.
343 PMLR, 2021.

A Implementation Details

This section illustrates the parameters chosen in the workflow. The parameters are classified into three categories: network architecture-related parameters, loss function-related parameters, and the approach to eliminate cyclic loops in the AIG netlist.

Design of the parameter in the Gumbel softmax function. The gumbel softmax function aims to select the connections among nodes, which has two additional parameters to set. (1) **tau**. The tau value denotes the function is more like a one-hot vector or a smoother one. Because the lower tau value causes a large gradient, further resulting in not diverging, we set the tau to 1. (2) **mode**. We set the mode to soft because soft mode can make probability information forward, while hard mode only passes 0/1.

Design of the loss function. A tailored loss function may improve the training accuracy. For a single-output bit logic network, the training stage backpropagates through the network when the current inference value differs from the expected value. For multiple-output bit logic networks, the training stage traverses every output bit and backpropagates only when the inferred bit value differs from the expected value.

x	y	z	n	o	
0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	1	
0	0	1	1	0	

```

case ( {x, y, z, n} )
  4'b0000: o<=0
  4'b0001: o<=1
  4'b0010: o<=1
  4'b0001: o<=0
  default: o<=0
endcase

```

Figure 10: The truth-table representation in Verilog format.

Initialization with Yosys tool. We implement the logic initialization with yosys, a kind of logic synthesis tool to process truth-table as an auxiliary option. The truth table can be represented as case statements in the Verilog program, which serves as the inputs of the Yosys tool. For example, Fig. 10 shows a case both in verilog case statement form and truth table format which are equivalent. In addition, because the yosys tool may generate a logic loop in the generated gate netlist, while the unified logic gate networks require networks without the loop. To eliminate the cyclic loop in the AIG netlist, we use `splitnets -ports` instruction in Yosys tool to eliminate the possible cyclic loop in the gate connections. The synthesis instructions in yosys can be represented in Fig. 11, where yosys first reads the Verilog file and then lowers it to AIG netlist.

B Additional Discussion of Differential Logic Networks Baselines

This section explores the differences among previous differential logic solutions by characterization them and providing an depth view on the differential logic.

Characterization of the Differential logic networks. Because the components in logic networks can be fixed or trainable, we can classify the differential logic networks from a component-trainable perspective. The unified representation can be classified into input trainable and logic node fixed type.

(1) Input connection fixed. As shown in Tab. 6, the "node fixed" column shows the loss function. It has a better convergence. However, this mode has a fixed connection and can not perform training thus, it can not directly be used in the logic synthesis process.

(2) Logic node fixed. The "Node Fixed" column in Tab. 6, which shows a diverging and smooth loss curve. This solution can better represent the connections change in the real-world circuit, which is chosen to be our baseline method.

```

1 read_verilog multiply.v
2 # check design hierarchy
3 hierarchy
4 # translate processes (always blocks)
5 proc
6 # detect and optimize FSM encodings
7 fsm
8 # implement memories (arrays)
9 memory
10 # convert to gate logic
11 techmap
12 aigmap
13 #eliminate the
14 write_json out.json
15 write_btor my.txt # truthtable
16 write_aiger -ascii out.aig

```

Figure 11: The yosys tool instructions in the multiply module synthesis example.

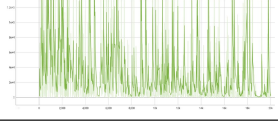

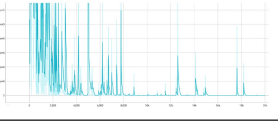
	Both Not Fixed	Node Fixed	Connection Fixed
Trainable	Node, Connection	Connection	Node
Fix	-	Node	Connection
Loss			
Paper	-	DeepMind-IWLS	DiffLogic-NeurIPS

Table 6: The loss function comparison of different fix modes. The test data is a simple 8-bit input logic formula.

(3) Both input connections and logic nodes are not fixed. The "Both Not Fixed" column of Tab. 6 shows a case for nodes and connections not fixing. The loss curve does not converge because too many factors in the models are not fixed. Specifically, the nodes and connections are all dynamic and can not perform good convergence.

The inefficiency of the differential logic transition function. We explore the logic transition function such as NAND ($1 - x \cdot y$). This assumption is not accurate because NAND can have different representations with the same forward but different gradient values such as $(1 - 0.5x \cdot y)$, which means the assumption is not unique. Further, this inaccurate may results in an error accumulation.

C Detailed Experiment of Logic initialization

This section provides the experiment details on IWLS benchmark, which consists of random initialization in Tab. 7 and initialized with the proposed initialization method in Tab. 8.

Table 7: Experiment details on IWLS benchmark (with the random initialization).

ex00	ex01	ex02	ex03	ex04	ex05	ex06	ex07	ex08	ex09
74.02%	60.23%	62.47%	69.14%	37.50%	47.31%	90.72%	37.50%	53.27%	79.56%
ex10	ex11	ex12	ex13	ex14	ex15	ex16	ex17	ex18	ex19
33.98%	41.65%	75.00%	31.25%	28.12%	20.90%	41.26%	76.56%	12.89%	58.20%
ex20	ex21	ex22	ex23	ex24	ex25	ex26	ex27	ex28	ex29
56.86%	98.05%	1.56%	70.31%	4.30%	51.54%	71.40%	61.13%	55.74%	44.17%
ex30	ex31	ex32	ex33	ex34	ex35	ex36	ex37	ex38	ex39
82.03%	69.99%	71.88%	61.30%	2.08%	53.91%	15.92%	84.40%	17.46%	21.88%
ex40	ex41	ex42	ex43	ex44	ex45	ex46	ex47	ex48	ex49
77.44%	68.75%	60.89%	61.72%	73.83%	24.83%	85.79%	13.09%	12.50%	75.98%
ex50	ex51	ex52	ex53	ex54	ex55	ex56	ex57	ex58	ex59
40.08%	17.43%	2.34%	82.03%	33.98%	60.35%	16.25%	72.66%	79.76%	12.90%
ex60	ex61	ex62	ex63	ex64	ex65	ex66	ex67	ex68	ex69
44.92%	89.06%	64.11%	28.61%	59.23%	25.00%	56.81%	58.13%	92.58%	17.19%
ex70	ex71	ex72	ex73	ex74	ex75	ex76	ex77	ex78	ex79
49.54%	10.94%	59.33%	67.85%	14.11%	64.84%	59.59%	82.54%	0.98%	42.19%
ex80	ex81	ex82	ex83	ex84	ex85	ex86	ex87	ex88	ex89
82.42%	16.99%	9.12%	28.32%	14.03%	81.30%	76.26%	56.84%	57.40%	61.13%
ex90	ex91	ex92	ex93	ex94	ex95	ex96	ex97	ex98	ex99
46.88%	72.79%	65.23%	51.82%	38.79%	34.38%	41.41%	32.23%	72.70%	62.50%

Table 8: Experiment details on IWLS benchmark (with the proposed initialization).

ex00	ex01	ex02	ex03	ex04	ex05	ex06	ex07	ex08	ex09
80.40%	87.08%	96.88%	88.97%	81.35%	86.18%	79.98%	96.88%	81.72%	80.00%
ex10	ex11	ex12	ex13	ex14	ex15	ex16	ex17	ex18	ex19
79.98%	80.00%	93.75%	88.67%	79.69%	79.98%	85.63%	83.98%	79.98%	92.19%
ex20	ex21	ex22	ex23	ex24	ex25	ex26	ex27	ex28	ex29
80.14%	93.41%	79.98%	99.22%	79.88%	90.00%	80.00%	88.98%	80.81%	84.85%
ex30	ex31	ex32	ex33	ex34	ex35	ex36	ex37	ex38	ex39
94.53%	80.00%	93.75%	82.53%	85.65%	91.41%	80.00%	80.00%	80.00%	89.84%
ex40	ex41	ex42	ex43	ex44	ex45	ex46	ex47	ex48	ex49
82.48%	89.06%	84.53%	88.36%	91.21%	80.91%	80.00%	79.98%	79.98%	92.75%
ex50	ex51	ex52	ex53	ex54	ex55	ex56	ex57	ex58	ex59
90.25%	82.35%	89.84%	94.43%	90.00%	95.48%	80.00%	98.83%	90.02%	80.00%
ex60	ex61	ex62	ex63	ex64	ex65	ex66	ex67	ex68	ex69
86.53%	83.52%	64.11%	79.98%	100.00%	79.69%	82.10%	81.82%	80.57%	80.00%
ex70	ex71	ex72	ex73	ex74	ex75	ex76	ex77	ex78	ex79
80.62%	81.84%	80.00%	80.12%	81.25%	80.05%	80.81%	80.00%	80.66%	86.72%
ex80	ex81	ex82	ex83	ex84	ex85	ex86	ex87	ex88	ex89
80.41%	84.77%	80.06%	79.98%	90.58%	80.33%	83.95%	81.88%	87.12%	85.16%
ex90	ex91	ex92	ex93	ex94	ex95	ex96	ex97	ex98	ex99
90.62%	83.15%	83.54%	80.00%	82.03%	82.03%	92.19%	87.70%	98.99%	89.06%

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [NA].
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes]" is generally preferable to "[No]", it is perfectly acceptable to answer "[No]" provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No]" or "[NA]" is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- Delete this instruction block, but keep the section heading "NeurIPS paper checklist",
- Keep the checklist subsection headings, questions/answers and guidelines below.
- Do not modify the questions and only use the provided macros for your answers.

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes] We accurately claim the contributions and scope.

Justification: [Yes] Sec. 1

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes] We have discussed the limitations.

Justification: [Yes] Sec. 7

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA] The paper does not include theoretical results.

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental Result Reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes] The paper provides the initialization converting code after yosys translating.

Justification: [Yes] Sec. 6.1

Guidelines:

- The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes] The released code is required to install yosys software.

Justification: [Yes] xxx

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes] We have provided with these information in this paper.

Justification: [Yes] Sec. 6.1

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes] We use the accuracy of every case to report error.

Justification: [Yes] Sec. 6

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes] We have provided sufficient information on the computer resources.

Justification: [Yes] Sec. 6.1

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines?>

Answer: [Yes] I have reviewed the Code of Ethics.

Justification: [Yes]

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA] Only for circuit design field, no societal impact.

Justification: [NA]

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA] A circuit model with no misuse risk.

Justification: [NA]

Guidelines:

- The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make the best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA] We do not use assets.

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New Assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes] The open-source code is on anonymous GitHub.

Justification: [Yes]

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and Research with Human Subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA] No crowdsourcing.

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.