CPP: A Multi-Level Circuit Partitioning Predictor for Hardware Verification Systems

Xinshi Zang
CSE Department, The Chinese University of Hong Kong
xszang@cse.cuhk.edu.hk

Lei Chen
Huawei Noah's Ark Lab, Hong Kong
lc.leichen@huawei.com

Xing Li
Huawei Noah's Ark Lab, Hong Kong
li.xing2@huawei.com

Wilson W.K. Thong
Huawei Hong Kong Research Center
thongwangkitwilson@huawei.com

Weihua Sheng
Huawei Hong Kong Research Center
sheng.weihua@huawei.com

Evangeline F.Y. Young
CSE Department, The Chinese University of Hong Kong
fyyoung@cse.cuhk.edu.hk

Martin D.F. Wong
CSE Department, The Chinese University of Hong Kong
mdfwong@cuhk.edu.hk

ABSTRACT

Circuit partitioning is a critical step in hardware-assisted functional verification that involves splitting a circuit into multiple partitions and assigning them to specific hardware. However, partitioning a large circuit can require considerable computation resources and time, especially when complex hardware constraints are involved. Moreover, the path delay after partitioning can have a significant impact on verification efficiency, making early path delay prediction crucial for refining the circuit effectively. In this work, we propose a novel circuit partitioning predictor, named CPP, to rapidly and accurately predict the path delay after partitioning. To achieve this, we use circuit coarsening to develop a multi-level path representation and employ a convolutional neural network (CNN) that can capture both local and global path structures for delay prediction. Through extensive experiments on large industrial circuits, we demonstrate the superiority of our prediction framework.

ACM Reference Format:
Xinshi Zang, Lei Chen, Xing Li, Wilson W.K. Thong, Weihua Sheng, Evangeline F.Y. Young, and Martin D.F. Wong. 2018. CPP: A Multi-Level Circuit Partitioning Predictor for Hardware Verification Systems. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Functional verification is a crucial step in electronic design automation (EDA) that ensures a logic design adheres to its specification before chip tapout. Due to the significant growth in modern System-on-Chip (SoC) and Application-Specific Integrated Circuit (ASIC) design size and complexity, the verification process often comprises 60-80% of the total development cycle [8]. To expedite circuit verification, hardware systems like multi-FPGA [12] and multi-processor systems [11] are developed to provide fast simulation platforms.

Circuit partitioning plays a significant role in standard compilation flow for verification platforms [5]. After logic synthesis, the Register Transfer Level (RTL) circuit is converted to a Look-Up-Table (LUT) based circuit, which is then partitioned into multiple partitions and assigned to specific hardware units. The maximum delay of the paths crossing different hardware units is the primary metric for evaluating partitioning in hardware systems [1]. Logic synthesis typically aims to minimize the netlist area and logic level without precise information on hardware systems [4]. Therefore, incorporating physical information, such as critical paths considering physical delays, into logic synthesis may optimize the netlist structure for better verification performance. However, partitioning a large circuit involves multiple optimization steps and is time-consuming due to complex hardware constraints. In practice, a fast and accurate method for estimating path delay is useful for identifying critical paths during partitioning.

In the field of physical design, deep learning methods such as CNN [7, 14], GNN, and Transformer [2, 15, 16] have been successfully applied to predict key metrics in various tasks, including net length and routability in placement and routing. For path delay prediction, Neto et al. [7] and Xu et al. [16] have borrowed concepts from Natural Language Processing (NLP) by considering logic gates as words and paths as sentences. They used Transformer [16] and CNN [7] models, respectively, to predict the path delay after synthesis and routing. However, there is currently no existing work on predicting path delay after partitioning in hardware systems. Compared with other predicting tasks, circuit partitioning prediction faces two challenges: (1) The input circuit is incredibly large and can contain millions or billions of nodes, making it impractical to apply existing prediction methods, such as GNN, directly. (2) The attributes of circuit nodes are not informative, as all logic gates are converted into standard LUT nodes in logic synthesis. As a result, the input LUT-based circuit for partitioning can be regarded as a plain graph without meaningful node attributes. Some previous approaches [7, 16] have not been able to achieve satisfactory performance in this problem due to the lack of node attributes.

To address the aforementioned challenges, we propose a novel learning-based Circuit Partitioning Predictor (CPP) to estimate the path delay after partitioning in hardware systems. The path sampling strategies used in [7, 16] either select paths randomly or select only the longest ones, which will usually need an extremely large amount of memory resources for a large-scale netlist. To overcome this limitation and extract representative paths more efficiently, we devise a practical diversity-driven path sampling strategy. Furthermore, we propose a novel multi-level feature representation

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

that takes into account both local and global structures in partitioning. By coarsening the netlist level by level, we encode high-level structures for each node and use a CNN model to learn a mapping from the multi-level path representation to the path delay. The contributions of this work are summarized as follows.

- We are the first to study and propose an effective deep learning method named CPP for predicting path delay after circuit partitioning applied in hardware verification systems.
- We propose an effective diversity-driven path sampling strategy that extracts representative and distinctive paths from large netlists for prediction and evaluation.
- We propose a novel multi-level path representation and utilize a CNN model to capture both local and global structures to predict the path delay after partitioning.
- We conduct extensive experiments on large industrial circuits to demonstrate the effectiveness and superiority of our proposed prediction method.
- We integrate our partitioning predictor into a logic synthesis tool. It can run 4× faster while achieving similar improvements on the maximum path delay compared to a real-world partitioning tool.

2 PRELIMINARIES

This section will cover several key concepts related to the LUT-based netlist. Additionally, we will describe typical verification hardware systems based on FPGAs and processors. Important terminologies are summarized in Tab. 1.

Table 1: Terminology

Term	Description
G(E, V)	Netlist graph; E is edge set, V is node set
NT	Node types, including input (IN), output (OUT), register (REG), and LUT
PI	Primary input set or beginning nodes set of paths, including all IN and REG nodes
PO	Primary output set or ending nodes set of paths, including all OUT and REG nodes
ID	Node input degree, i.e. the number of incoming edges
OD	Node output degree, i.e. the number of outgoing edges
D	Node total degree, $D = ID + OD$
WD	Node total weighted degree, i.e. the sum of edge weights
OP[i], ON[i]	The number of outgoing paths and neighbors of node i

2.1 LUT-Based Netlist

Because a k-input Look-Up-Table (LUT) can represent any boolean expression with k variables, both FPGA-based and processor-based hardware systems use LUTs to simulate the logic functionality. After logic synthesis, the RTL netlist is converted to a LUT-based netlist with all logic gates replaced by LUT nodes. In this work, except for LUT nodes, we also consider three other kinds of nodes including IN, OUT, and REG. REG nodes work as sequential components in a netlist and the beginning and ending nodes of timing paths. There are four kinds of timing paths in a netlist: IN-to-REG, IN-to-OUT, REG-to-REG, and REG-to-OUT. Fig. 1 shows an example of a LUT-based sub-netlist. There are in total 27 timing paths starting from PI to PO, which is equal to the sum of OP of PI. The OP calculation for each node will be explained in Sec. 3.1.

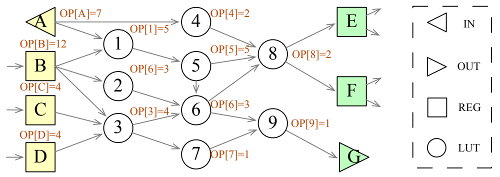

Figure 1: A LUT-based sub-netlist. Yellow and green shapes denote the PI and PO respectively.

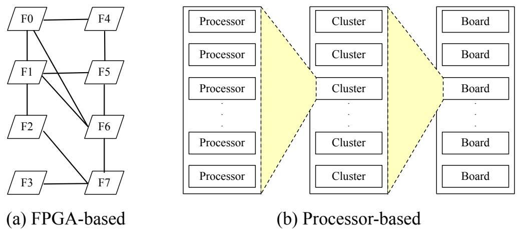

Figure 2: Examples of verification hardware systems.

2.2 Hardware Verification Systems

Commercial hardware-assisted verification systems are typically composed of either multi-FPGAs or processor arrays, as shown in Fig. 2 [10, 11]. Unlike timing analysis in the physical design of ASICs, the delay of LUT nodes in these verification systems can be assumed to be the same and small. Similarly, the net delay within one FPGA or processor is also assumed to be the same and small. However, the communication delay between two FPGAs or processor boards is usually larger than that within a single FPGA or processor. The maximum path delay generally determines the frequency at which a verification system can operate. Note that these hardware systems are used only for functional verification and do not affect the original circuit design's timing.

3 METHODOLOGY

In this section, we present our circuit partitioning predictor (CPP) for verification systems. The framework of CPP is illustrated in Fig. 3. To facilitate path delay prediction, we first propose a diversity-driven path sampling algorithm to extract sufficient representative paths from the LUT-based netlist. We then use circuit coarsening to obtain a multi-level path representation that captures the structural features of circuit nodes on different levels. The multi-level path features are represented in a standard image format, which allows us to train a simple yet efficient CNN model to learn a mapping from the input features to the path delay.

3.1 Diversity-Driven Path Sampling

Different from timing optimization in ASICs, the LUT-based netlist in verification systems can just be regarded as an ordinary hypergraph without timing constraints. The path delay can only be determined after the netlist is partitioned to hardware systems. Therefore, path coverage and diversity are the major concerns for any before-partitioning path sampling strategy.

In a LUT-based netlist, the number of paths can be estimated as NK^L, where $N = |PI|$, K is the average OD, and L is the average path length. For a large netlist, the path number could be uncountable due to the large values of N and L. Moreover, many of these paths share common nodes, making them similar to each other. For

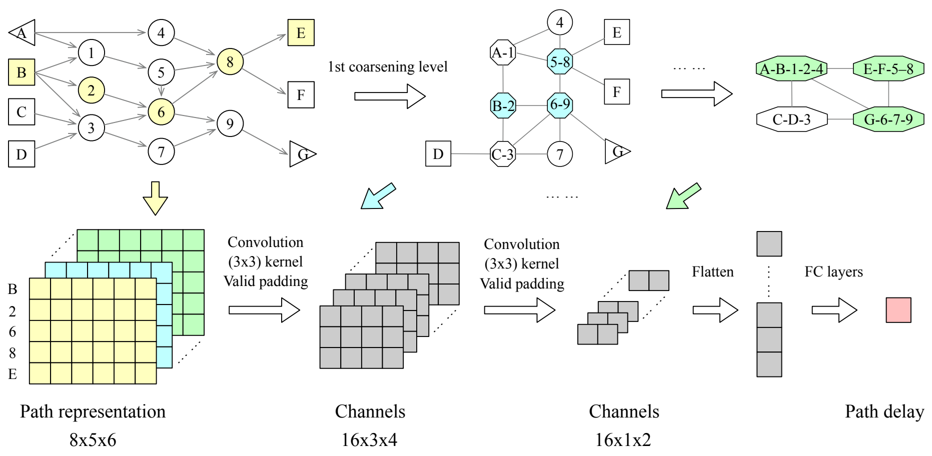

Figure 3: CPP framework. The top row represents the circuit coarsening procedure and multi-level feature extraction. The bottom row shows the CNN model to predict path delay based on the multi-level path representation.

example, the paths `A-1-5-8-E` and `A-1-5-8-F` in Fig. 1 are similar. Therefore, an efficient path sampling strategy is necessary to extract representative and distinctive paths from a netlist.

The path sampling strategies proposed in [7] and [16] suffer from scalability issues when applied to large netlists. For instance, Neto et al. [7] enumerate all longest paths between each $\langle pi, po\rangle$ pair, resulting in a sampled path number of approximately NM, where M is the average number of $\langle pi, po\rangle$ pair for each pi. This approach can create bias by overlooking short but critical paths after partitioning. On the other hand, Xu et al. [16] utilize a random sampling strategy that performs a depth-first search from PI and only randomly selects k ($k \ll K$) outgoing nodes each time. Although this reduces the maximum sampled path number to Nk^L, it could still be exponential when L increases. Furthermore, this method can generate many similar paths due to the random selection of outgoing nodes, which can result in many nodes with large IDs being picked many times.

To address the limitations of the existing sampling methods, we propose a diversity-driven path sampling algorithm shown in Alg. 1. Because the OP reflects the pi's connectivity and importance on path sampling, we first randomly pick a portion of PI based on the distribution of OP. The larger OP[pi], the higher the possibility of pi being selected. The OP for each node is calculated in dynamic programming using Eq. (1). We set the initial OP values for each po to 1. The complexity of calculating OP is $O(|E| + |V|)$.

$$OP[v_i] = \sum_{v_j \in ON[v_i]} OP[v_j] \tag{1}$$

Starting from selected pis, we perform a diversity-driven sampling strategy to recursively construct paths (line 8). We select nodes with the largest ID first (line 12 and 13) since they have more outgoing paths passing through them. At each iteration, we prioritize selecting the longest outgoing path from the current node being visited (line 16). We use the stopping status to prune out nodes without feasible outgoing paths (line 19). Once a path is sampled, we update the IDs of the nodes along the path to reduce the possibility of sampling similar paths later (line 6).

The complexity of the path sampling in Alg. 1 is $O(|E| + |V|)$. The sampled path number is around nm, where m is the average OD of PI. Compared with [7, 16], the number of sampled paths is considerably fewer and the path similarity is well controlled.

Algorithm 1: Diversity-driven Path Sampling

```
Input: Netlist; Output: Path set: $PS$
1  $PI' \leftarrow$ randomly sample $n$ unique $pi$s from $OP$ distribution
2  $stop \leftarrow false$ for each node
3  for $pi \in PI'$ do
4      $p \leftarrow$ sample($pi$)
5      Add $p$ into $PS$ if $p$ not empty
6      $ID[v] \leftarrow ID[v] - 1$ for each $v \in p$
7  Go to line 3 until no more path can be found
8  Function sample($v_i$)
9      $p \leftarrow \{\}, p' \leftarrow \{\}$
10     $V_1 \leftarrow \{v | v \in ON(v_i), v \in PO, ID[v] \neq 0, !stop[v]\}$
11     $V_2 \leftarrow \{v | v \in ON(v_i), v \notin PO, ID[v] \neq 0, !stop[v]\}$
12     $v_j \leftarrow v \in V_1$ && $ID[v]$ is max
13     $v_c \leftarrow v \in V_2$ && $ID[v]$ is max
14     $p' \leftarrow$ sample($v_c$)
15     if $p'$ is not None then
16         $p \leftarrow \{v_i\} + p'$
17     else if $v_j$ is not None then
18         $p \leftarrow \{v_i, v_j\}$
19     $stop[v_i] \leftarrow true$ if $v_j$ is None && $p'$ is None
20     return $p$
```

3.2 Multi-Level Path Representation

Because most nodes in a LUT-based circuit are general and uniform LUTs, their functionality does not significantly affect the final partitioning results. Instead, the local and global structures of the nodes play a more important role in partitioning. Currently, popular partitioning tools for general hypergraphs, like hMETIS [3] and KaHyPar [9], adopt a multi-level framework consisting of coarsening, initial partitioning, uncoarsening, and refinement. This multi-level approach effectively handles the global structure of the hypergraph and minimizes the cut size.

To capture both local and global structures, we propose a multi-level path representation based on circuit coarsening inspired by the multi-level partitioning framework. As illustrated in Fig. 3, our approach involves a heavy edge matching strategy that merges

Table 2: Examples of base-level and first-level representation for path `B-2-6-8-E`.

Nodes	Base-Level						First-Level					
	IN	OUT	REG	LUT	ID	OD	IN	OUT	REG	LUT	D	WD
B	0	0	1	0	0	3	0	0	1	1	3	3
2	0	0	0	1	1	1	0	0	1	1	3	3
6	0	0	0	1	3	2	0	0	0	2	5	6
8	0	0	0	1	3	2	0	0	0	2	5	6
E	0	0	1	0	1	0	0	0	1	0	1	1

nodes connected with the net whose weight is the largest into a new super-node. The net weight reflects the number of nets being merged, and no super-node can be merged again. Once there are no more nodes to merge, we obtain the first-level coarsened graph. We can apply similar procedures to obtain many coarsened graphs of different levels. However, the coarsened graphs are undirected because the edge direction cannot be preserved during coarsening.

After obtaining multiple coarsened graphs, we construct the multi-level representation for each path. The base-level node feature includes the one-hot encoding node type, ID, and OD on the original netlist. Additionally, we prepare features for seven other coarsened graphs, namely, the 1st to 5th, 8th, and 16th levels. We select these low-level coarsened graphs to provide more distinctive features for nodes in a path. These high-level coarsened graphs are used to extract global structures for paths. Like the base-level node features, the other-level node features consist of the node number for each node type, D, and WD of its super-node. An example of base-level and first-level path representation is shown in Tab. 2. Because of the different scales of node features, we will conduct standard normalization on each level separately.

3.3 Prediction Model

The multi-level path representation can intuitively be regarded as a standard image, where the node features in the same path are spatially correlated, and so are the path features in different levels. Therefore, we utilize a CNN model to capture these local and global features and embed neighboring nodes in one path. As shown in Fig. 3, the CNN model contains two convolutional layers with $16 \times 3 \times 3$ kernels and two fully-connected layers with 256 and 128 neurons, respectively. The kernel size is set to 3 because there are only three kinds of node features related to the node type and degrees. This CNN model will learn a mapping from the multi-level path representation to the path delay. Necessary zero padding is applied to make the CNN input have the same shape.

4 EXPERIMENT

In this section, we first set up the experiments by introducing netlist datasets, baselines, and evaluation metrics. Then, we focus on discussing the prediction performance of different methods. Finally, we demonstrate an application of path delay prediction in logic re-synthesis. We implement the path sampling algorithm and multi-level feature extraction in C++, while the prediction model is implemented in Python. All experiments are conducted on a server with 72 CPU cores and one NVIDIA A100 GPU.

4.1 Datasets

As the open-source circuit benchmarks used in related works [7, 16] are not large enough for the hardware-assistant verification scenario, we utilize 20 large LUT-based netlists from industry, whose

Table 3: LUT-based circuit statistics. M and K stand for million and kilo respectively.

Statistic	#Node (M)	#Edge (M)	#Path (K)
Minimum	1.07	1.08	62.36
Average	10.60	10.73	148.39
Maximum	39.90	40.43	367.30

statistics are summarized in Tab. 3. Like [15], we ignore those paths whose length is over 100 to prevent an extraordinarily large range of path delay and to reduce computation overhead in length padding. Compared with other paths, the number of these very long paths can be neglected. Finally, there are around 2.97 million paths extracted from these netlists. The path delay labels are generated using a partitioning tool. The path set is split with a 60%-20%-20% ratio for training, validation, and testing respectively.

4.2 Baselines

We compare CPP with three methods explained as follows.

(1) WLM: Inspired by the wire load model (WLM) [13] for wire delay estimation, we propose a heuristic method by calculating the edge delay based on the OD of the sink node. If $OD \in (10^{n-1}, 10^{n}]$, the edge delay is estimated with $n+1$. The estimated path delay is then obtained by accumulating the edge delays along a path.
(2) EaSyOpt [7]: This method uses CNN to classify critical paths after routing. Cycle time and node output load are two primary input features for this model, which do not exist in the partitioning problem. To make a fair comparison, the base-level path features are used for this method. The model hyper-parameters follow its released implementation.
(3) Circuitformer [16]: This method utilizes Transformer to predict path timing for synthesis and the node type is the only node feature. As node types are not discriminative anymore in LUT-based netlist, similarly the base-level path features are also used as the input features for this method. The model hyper-parameters follow what is suggested in [16].

For all learning-based methods, the models are trained with the same Adam optimizer, mean square error (MSE) as the loss function, and 1000 training epochs. The best models on the validation dataset are evaluated on the testing dataset.

4.3 Evaluation Metrics

Following the evaluation metrics in [16], we use the Root Relative Squared Error (RRSE) and Mean Absolute Percentage Error (MAPE) to evaluate prediction performance. As defined in Eq. (2), d, d' and $\overline{d}$ represent the real delay, predicted delay, and the mean of real delays respectively. RRSE scales the root mean square error by the variance of the ground truth and MAPE is the mean percentage error by which the prediction differs from the ground truth.

$$RRSE = \sqrt{\frac{\sum_{i=1}^{n}(d_i - d_i')^2}{\sum_{i=1}^{n}(d_i - \overline{d})^2}}, \quad MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|d_i - d_i'|}{d_i} \tag{2}$$

4.4 Prediction Performance

The overall prediction results are shown in Tab. 4. The RRSE of WLM and the MAPE of Circuitformer are the worst, revealing that Transformer may fail to learn useful knowledge without meaningful

Table 4: Overall prediction performance

Metric	WLM	EaSyOpt [7]	Circuitformer [16]	CPP
RRSE	0.90	0.58	0.73	**0.53**
MAPE	44.73%	36.27%	59.64%	**28.57%**

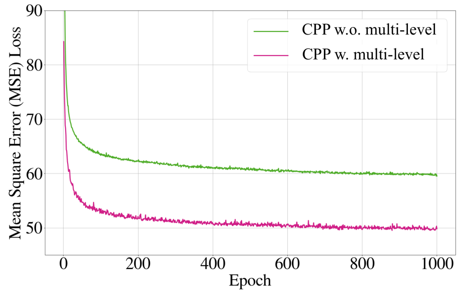

Figure 4: Validation curves of CPP with and without multi-level features.

node attributes. EaSyOpt performs the best among those baselines. CPP can further outperform EaSyOpt on both RRSE and MAPE by 8.62% and 21.23% respectively, showing the superiority of our proposed framework. Owing to our diversity-driven path sampling, the variance of path delays can be high, which also explains why the improvement on MAPE is larger than that on RRSE.

4.5 Ablation Study

To investigate precisely the influence of our multi-level path representation, we conduct an ablation study by only using basic path features, i.e., the base-level path representation in Fig. 3 for CPP. In this way, the major difference between CPP and EaSyOpt only reflects on the model hyper-parameters. The RRSE and MAPE of CPP without multi-level features are 0.58 and 34.13% respectively. Therefore, the multi-level path representation indeed helps CPP capture more global structure features and contributes a lot to the prediction accuracy. Furthermore, the validation curves during the training stage are drawn in Fig. 4. It can be seen that with the multi-level features, CPP can achieve a better and faster convergence.

4.6 Application in Logic Re-Synthesis

We also explore the application of circuit partitioning prediction in logic re-synthesis. To this end, we propose a novel re-synthesis strategy that takes into account the critical paths identified during partitioning. First, the paths whose delays are among the top 10% of all sampled paths in a netlist are regarded as critical paths. We then apply lutpack, an area-oriented re-synthesis engine in ABC [6], to solely optimize combinational nodes along critical paths. At last, partitioning is performed on the re-synthesized netlist.

We compared two methods for selecting critical paths: using a reference partitioning tool to obtain the real path delay, and using CPP to predict the delay. We evaluated these methods on two LUT-based netlists that were not used in the experiments in Section 4.1, and which were therefore unseen by CPP. As shown in Tab. 5, compared with the original partitioning results, the re-synthesis with predicted critical path feedback further reduce the maximum path delay in the hardware system to 65% and 88%, respectively. Compared with the real partitioning tool, CPP can generate the critical paths nearly four times faster and help the re-synthesis to achieve similar improvements on the maximum path delay. The

Table 5: Re-synthesis and partitioning results. Runtime is the time to obtain critical paths. The runtime format of the predicted is feature extraction time + prediction time.

Netlist	#Node (M)	#Edge (M)	Max. Delay Ratio		Runtime (s)		
			Real	Predicted	Real	Predicted	Ratio
Netlist1	17.35	17.38	0.61	0.65	273	61+5	0.24
Netlist2	26.62	26.67	0.84	0.88	398	88+7	0.24

prediction time of CPP is very fast and most of the running time is spent on feature extraction.

5 CONCLUSION

In this work, we propose a novel circuit partitioning predictor (CPP) to predict the path delay in verification hardware systems. Our scalable path sampling strategy and multi-level path representation, combined with a CNN model, allowed us to capture different levels of path features and achieve superior prediction accuracy compared to existing methods. Moreover, our approach showed great potential for reducing critical path prediction runtime in logic re-synthesis.

REFERENCES

[1] Ming-Hung Chen, Yao-Wen Chang, and Jun-Jie Wang. 2021. Performance-driven simultaneous partitioning and routing for multi-FPGA systems. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1129–1134.

[2] Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z Pan, and Yibo Lin. 2022. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference.*

[3] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the 34th annual Design Automation Conference.* 526–529.

[4] Xing Li, Lei Chen, Fan Yang, Mingxuan Yuan, Hongli Yan, and Yupeng Wan. 2022. HIMap: a heuristic and iterative logic synthesis approach. In *Proceedings of the 59th ACM/IEEE Design Automation Conference.* 415–420.

[5] Marisa López-Vallejo and Juan Carlos López. 2003. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 8, 3 (2003), 269–297.

[6] Alan Mishchenko, Robert Brayton, and Satrajit Chatterjee. 2008. Boolean factoring and decomposition of logic networks. In *2008 IEEE/ACM International Conference on Computer-Aided Design.* IEEE, 38–44.

[7] Walter Lau Neto, Matheus Trevisan Moreira, Luca Amaru, Cunxi Yu, and Pierre-Emmanuel Gaillardon. 2021. Read your circuit: leveraging word embedding to guide logic optimization. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference.* 530–535.

[8] Michael Santarini. 2005. *ASIC prototyping: make versus buy.* https://www.edn.com/asic-prototyping-make-versus-buy

[9] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2016. K-way hypergraph partitioning via n-level recursive bisection. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX).* SIAM, 53–67.

[10] Yu-Hsuan Su, Richard Sun, and Pei-Hsin Ho. 2019. 2019 CAD contest: System-level FPGA routing with timing division multiplexing technique. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE.

[11] Ray Turner. 2004. *A primer on processor-based emulation.* https://www.eetimes.com/a-primer-on-processor-based-emulation/

[12] Joseph Varghese, Michael Butts, and Jon Batcheller. 1993. An efficient logic emulation system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 1, 2 (1993), 171–174.

[13] A Windschiegl, P Zuber, and W Stechele. 2002. A wire load model for more accurate power estimation. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, Vol. 1. IEEE, I–376.

[14] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE, 1–8.

[15] Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, Jiang Hu, Yixiao Duan, and Yiran Chen. 2021. Net2: A graph attention network method customized for pre-placement net length estimation. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference.* 671–677.

[16] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. 2022. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In *Proceedings of the 49th Annual International Symposium on Computer Architecture.* 847–859.