Lightweight Structural Choices Operator for Technology Mapping

Antoine Grosnit[*]
Huawei Noah's Ark Lab
antoine.grosnit2@huawei.com

Matthieu Zimmer[*]
Huawei Noah's Ark Lab
matthieu.zimmer@huawei.com

Rasul Tutunov
Huawei Noah's Ark Lab
rasul.tutunov@huawei.com

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

Lei Chen[†]
Huawei Noah's Ark Lab
lc.leichen@huawei.com

Fan Yang
Huawei Noah's Ark Lab
yangfan40@huawei.com

Mingxuan Yuan
Huawei Noah's Ark Lab
yuan.mingxuan@huawei.com

Haitham Bou-Ammar
Huawei Noah's Ark Lab
University College London
haitham.ammar@huawei.com

***Abstract*—Technology mapping quality heavily depends on the subject graph structure. To overcome structural biases, operators construct choice nodes to enable mappings with improved node and level counts. Nevertheless, state-of-the-art structural choice operators scale poorly with graph size.**

We present the lightweight structural choices (LCH) operator that incorporates equivalencies by processing only subparts of the graph. We propose multiple heuristics that rely on specific node extraction orders and subpart sizes to extract non-overlapping components. Compared to state-of-the-art methods on EPFL circuits, LCH is 2.35x faster enduring a small sacrifice in node count (3%) and level reduction (2%).

***Index Terms*—logic synthesis, structural bias problem, structural choice operators**

I. Introduction

Combinational technology mappers [1] convert a given Boolean network into a functionally equivalent network made of library gates, while optimising certain quality metrics such as total area or delay. This problem is known to be NP-hard [2], and researchers simplify it by restricting the mapped network to be structurally similar to the subject network by framing it as a covering problem [3], [4]. To be successful, this approach requires the subject graph to have a "good" underlying structure, which can typically be obtained by applying logic optimisation operators before performing technology mapping. However, there is a gap between the quality metrics measured on the optimised subject graph and those measured on the mapped netlist. Specifically, a well structured subject network will not always lead to a high quality mapped networks [5]. As a consequence, the covering-based methods can miss the best overall solution even if we optimally solve each of logic optimisation and covering steps.

To mitigate the bias induced by the subject graph structure on the technology mapping quality, Chatterje et al. propose lossless logic synthesis [6]. Rather than using a single optimised subject graph as input to the mapping, lossless synthesis combines Boolean networks with different structures and leaves the selection of the best parts of each network to the mapper. The different structures come from the intermediate networks observed at different stages when applying logic optimisation flows to the subject graph.

Collecting structural choices helps to reduce structural bias and to achieve better mapping results [6]–[8]. Although successful, these methods also share a drawback hindering them from many practical applications with large circuits. Indeed, proving or disproving the equivalence of internal nodes in the entire subject network might require calling SAT solvers on large parts of the subject network. Although advanced methods such as fraiging, simulation and re-simulation [9] accelerate the equivalence check, they can still be too slow to be executed on large network instances, which makes lossless synthesis scale poorly with the size of the input subject graph.

To improve the efficiency of choice network construction, we propose a lightweight choice generation method. Instead of processing the whole subject graph at once, we sequentially extract subgraphs and apply area-oriented logic optimisation flows followed by sweeping to get structural choices. The resulting subgraphs along with the new choices are then incorporated into the subject graph, and the process is repeated. Due to the exponential time complexity of the sweeping, running this operation on smaller subgraphs sequentially typically gives faster performance compared to executing sweeping on the whole subject graph.

The main contributions that we make in this paper are summarised as follows:

- a lightweight operator, *lch*, reducing the overall time complexity by calling SAT solvers on smaller subgraphs while allowing us to still build numerous structural choices,
- a non-overlapping strategy to select root nodes and extract subgraphs to process,

[*] Equal Contribution
[†] Corresponding Author

- an efficient heuristic to extract meaningful subgraphs depending on whether the starting node lies on a critical path of the subject graph or not,
- a thorough empirical evaluation on EPFL *Arithmetic* and *Random/Control* benchmarks to evaluate the proposed method.

The rest of the paper is organised as follows. Section II introduces background elements and concepts needed to understand the problem we tackle. In Section III, we present the framework of the proposed lightweight choice generation method. Experimental results are reported in Section IV, and Section V concludes the paper.

II. Background

A. Boolean network

In logic synthesis, a Boolean network is a Directed Acyclic Graph (DAG) representing a Boolean function usually characterised in hardware description languages [10]. A Boolean function $f : \{0,1\}^{d_i} \rightarrow \{0,1\}^{d_o}$ taking d_i variables as input and driving d_o outputs can be mapped into a Boolean network with d_i primary input nodes (PIs) and d_o primary output nodes (POs). PIs have no incoming edge, while POs have no outgoing edge and serve as entry points to other networks in the circuit environment. Inner nodes correspond to technology-independent gates (e.g. AND, XOR, etc.) [11] driven by some *fanins* connected to them by incoming edges, and driving some *fanouts* connected to them by outgoing edges.

The *level* of a node x is the length of the longest path from any PI to x, and the *level* of the Boolean network is given by the largest inner node level. A directed path from a PI to a PO and the inner nodes lying on such path are said *critical* if the length of this path is equal to the network level. The *area* of a Boolean network refers to the number of inner nodes present in the graph.

A *transitive fanin/fanout cone* (TFI/TFO) rooted at a node x is a set of nodes reachable through the fanins/fanouts of x and forming a connected subgraph. A *window* around a *pivot* node x is a set of nodes made of the union of a TFI and a TFO rooted at x and which forms a connected subgraph.

A cut C of a root node x is a set of nodes - known as *leaves* - such that all directed paths from a PI to x goes through one of the cut leaves. A cut is said K-feasible if its cardinality $|C|$ is less than or equal to K. The level of a cut is obtained by incrementing the largest leave level by one.

Numerous applications [11]–[13] rely on And-Inverter Graph (AIG) [14], a representation form amenable to efficient logic optimisation based on the aforementioned notions.

B. And-Inverter Graph

An AIG is a homogeneous type of Boolean network whose inner nodes represent two-input AND gates. An AIG edge can be marked to signify that the corresponding fanin signal is complemented before being passed to the driven AND gate. Any general Boolean network can be converted to an AIG by factoring its logic gates to obtain two-input AND/OR nodes, and by using DeMorgan's rule to transform disjunctions into conjunctions. Many logic operators have been designed to optimise the structure of the AIG, notably to reduce its area, such as *refactor*, *rewrite* [15], [16], *resub* [11], [17], or to achieve level reduction, such as *balance* [11], *SOP balance* (SOPB) [18] and *Lazy man's synthesis* (LMS) [19].

To obtain a simplified AIG with optimised area and level before performing technology mapping, a sequence of operators can be applied to the *subject graph*. For instance, logic optimisation flow *dresyn2* [16], included in the ABC package [20], targets area reduction with limited level increase by interleaving balancing, rewriting and refactoring.

C. Technology Mapping

A technology mapper takes as input a subject graph and produces a mapped netlist, which is a Boolean network whose nodes are actual library gates, such as standard cells for *application specific integrated circuits* (ASICs) or K-input *Look-Up Tables* (K-LUT) for FPGA-based designs. To get a mapped netlist realising the same Boolean function as the subject graph, the standard procedures consist in covering its nodes by library gates realising the same Boolean function. The quality of the technology mapping can be measured in terms of area, delay, testability, or power consumption, etc. In our experiments we will focus on FPGA mapping and measure the quality of results based on the number of K-LUTs (LUT-count) in the mapped netlist and the length of the longest path from a PI to a PO (LUT-level).

The problem of getting optimal mapped netlist through the covering procedure is highly complex due to its combinatorial nature. Heuristics based on fast enumeration of cuts and priority cuts have notably been designed [21] and refined [12], [13] to go through the most promising gate covering achievable for a given subject graph.

Nevertheless, the overall covering quality depends on the actual matching possibilities between the library gates and the graph structures present in the Boolean network. Therefore the quality of the mapped netlist is biased by the structure of the subject graph [7] and two subject graphs representing the same Boolean function can lead to very different quality of results.

One approach to mitigate this problem consists in augmenting the technology mapping library with *supergates* [6] that are single-output components recursively obtained by combining library gates together. This augmentation allows to find more advantageous matching as the gates combined into the supergates could not necessarily cover a portion of the structure covered by the supergate. An orthogonal approach consists in running logic optimisation operations directly on the mapped netlist softening the impact of the pre-mapping stage on the quality of results. Post-mapping operators include SAT-based area recovery (&satlut) [22] and *GSRW* [23] which iteratively extract, optimise and replace windows from the input mapped netlist, a procedure known as *resynthesis*. A third strategy that is compatible with the use of supergates and post-mapping refinement, tackles the structural bias problem at its source by enriching the subject graph structure through the addition of choice nodes [6], [7].

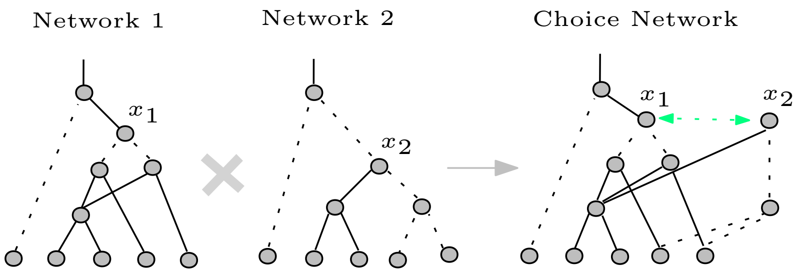

Fig. 1. Combine Networks to Generate a Choice Network

D. Choice networks

Two distinct nodes x_1 and x_2 of a Boolean network are equivalent (up to complementation) if, for any assignment of the PIs, the Boolean signal driven by x_1 is always the same (or always the complement) of the signal driven by x_2. When a subject graph contains two structurally different nodes that are marked as equivalent, cut-based technology mappers can consider the different covering alternatives to decide which of the two equivalent structures to cover, which often leads to higher-quality results [12].

As shown in [6], *lossless synthesis* applied to a subject graph G_0 allows to construct a choice network with inner nodes encoding the same Boolean function through different structures. The procedure starts by applying $\ell \geq 1$ logic optimisation flows to G_0 in order to obtain $G_1, ..., G_\ell$ Boolean networks equivalent to G_0 by construction. The second step involves merging the set of graphs $\{G_i\}_{0 \leq i \leq \ell}$ together and sweeping the resulting graph to identify structural choices. Figure 1 gives an example of the merge of two equivalent networks, resulting in a "Choice Network" where nodes x_1 and x_2 realise the same function. To identify structural choices, SAT-sweeping [6] is performed, gradually splitting the nodes into equivalence classes by running simulation and calling SAT-solver instances. Despite efforts of the community to improve the efficiency of SAT-sweeping [24]–[26] by exploiting counter-examples or designing circuit-based SAT solvers, the scalability of the sweeping operation remains limited as the underlying problems it solves is still NP-hard.

To address the scalability challenge posed by the global lossless synthesis approach, we propose a lightweight solution to node construction, based on subgraph synthesis and sweeping.

III. LCH operator

LCH (Algorithm 1) runs a loop composed of the following operations: it extracts a subgraph, performs logic synthesis operations, and sweeping to find structural choices within it. The resulting graph is then merged in the subject graph. This main loop iterates over every node of the subject graph in a topological order (from the PIs to the POs). To limit the number of extracted subgraphs, we define two hyperparameters: M is the maximum number of critical nodes from which an extraction can occur and P is the probability of extracting

Algorithm 1 LCH: Lightweight Structural Choices

```
Input: a subject AIG graph $G$, the maximum cut size $C$, the maximum number of nodes in a subgraph $N$, the maximum number of critical nodes to process $M$ and the probability of non-critical nodes to process $P$.
Output: A choice network equivalent to $G$
Identify the critical path of $G$
$m \leftarrow 0$
for each node $n$ in $G$ in topological order do
    if $n$ has already been included in previous subgraphs
        Skip $n$.
    end if
    if $n$ is critical
        if $m > M$
            Skip $n$.
        end if
        Extract a subgraph $D$ from $n$ using BFS multi-outputs with at most $C$ cuts and $N$ nodes.
        $m \leftarrow m + 1$
    else $n$ is not critical
        Skip $n$ with probability $1 - P$
        Extract a cone $D$ from $n$ using BFS with at most $C$ cuts and $N$ nodes.
    end if
    $D' \leftarrow$ Optimise $D$ with dresyn2.
    $D^\dagger \leftarrow$ Merge $D$ with $D'$.
    $D^* \leftarrow$ Sweep over $D^\dagger$.
    Merge $D^*$ into $G$.
end for
```

a subgraph from a non-critical node. We now present the extraction and the sweep operations in more details.

A. Subgraphs Extraction

To limit the size of the extracted subgraphs, we define two other hyperparameters: C the maximum number of nodes in the cut and N the maximum number of nodes in the subgraph. Given a node in the subject graph, we rely on two different strategies to extract a subgraph.

- If the starting node is not in the critical path of the subject graph, we start by extracting a subgraph from the current node in Breadth First Search (BFS) order defining the current node as a PO of the subgraph. At most N nodes

will be extracted with C PIs. Nodes that were already processed or been included in previous subgraphs are not considered.
- When the starting node is in the critical path, the strategy is similar except that we do not directly stop the construction if there are no more fanins to visit. We will visit the fanout of the oldest inserted node in the subgraph and treat it as the current node, repeating the process until N nodes are extracted, or C PIs are extracted, or no more non-visited fanins and fanouts are available.

To keep the future merge with the subject graph consistent, all nodes in the extracted subgraph that have fanouts outside of the subgraph are linked with an artificial PO in the subgraph. This ensures that these nodes will still be present for the later merge.

B. Logic Synthesis & Sweeping

Given the extracted subgraph D, we optimise it with the *dresyn2* sequence (see Section II-B). We denote the optimised version by $D^\dagger$. To merge D with $D^\dagger$ into D^*, the structural hashing of nodes is used to reduce redundant nodes (i.e. if a node with the same two fanins is already present it is reused). The merge is performed through a Depth-First Search (DFS) order from the primary input to the primary outputs. The resulting subgraph $D^\dagger$ have the same number of primary input as D but twice the number of primary outputs. Then, the sweep over $D^\dagger$ is obtained via calls to SAT solvers [6]. The merge of the choices found by the sweep in the subject graph is also done via DFS order with structural hashing. This last operation can increase the number of nodes in G by providing alternatives, but can also reduce it if simplifications are found.

In the worst case, if the verification was done on every node, the complexity of SAT solvers for the sweep would be proportional to $\mathcal{O}(n \times 2^k)$ where k is the number of primary inputs of a graph, and n is the number of nodes. It is notably the case for the other structural choice operators [6] that process the entire subject graph. Assuming that the subject graph could be decomposed in ℓ subgraphs with C being the maximum cut size, the complexity of LCH in the worst case would be dominated by $\mathcal{O}(n \times 2^C)$ such that $C \ll k$. Note that in practice the verification is not necessarily done on every node.

IV. Numerical Evaluations

To evaluate the effectiveness of our proposed LCH method, we implement and integrate it in the synthesis and verification system ABC [20]. The circuits used for the comparison come from the EPFL benchmark [27]. It is composed of two groups of circuits: arithmetic ones and random/control ones. We aim to accelerate the lossless logic synthesis [6], denoted DCH in ABC, while being better than doing no operation at all. To do so, we compare the following 3 sequences:

- DCH: *strash; dch; if -K 6*
- LCH: *strash; lch; if -K 6*
- none: *strash; if -K 6*

The *if* command in ABC performs mapping using the priority cut based algorithm [13]. It is capable of taking advantage of the choices present in the network. Note that on this set of circuits, vanilla DCH (without -l, -x, or -t) performs the best. We only report the time taken by LCH and DCH, the mapping time is not included nor the time for loading the graph in memory and strashing it. Since LCH is stochastic, we use 3 random seeds and report the arithmetic average of nodes, levels, and time. The experiments are carried out on a computer with Intel Core E5-2699 v4 @ 2.20GHZ. Each sequence is restricted to use only a single thread.

To tune the hyperparameter of LCH, we rely on multi-objective Bayesian optimisation [28]. To showcase generalisation, the tuning is done only on the 10 arithmetic circuits. The two objectives are the geometric average of time and node ratios with DCH. We manually pick up a point from the Pareto front with less speed improvement but small sacrifice in node and level reductions. It is $C = 165$, $P = 0.6$, $M = 94$ and $N = 31230$. Then, we obtain the performance of this point on random/control circuits of the EPFL benchmark.

The results are reported in Table I. Since we are comparing ratios, we reported the geometric average in the last row. On average, LCH is 2.35x faster than DCH, while enduring only a small sacrifice of 3% in node reduction and 2% in level reduction. It is also better than doing no operation at all by 7% in node reduction and 11% in level reduction.

We also report more details in terms of the number of generated choices and processed nodes in Table II. As expected, LCH generates less choices than DCH (60% less on average). However, it is able to provide more choices for larger circuits.

On average, LCH only processes 89% of the nodes, which can explain the speedup with the fact that the sweep operations are performed on smaller graphs. We also observe that on small circuits like bar, sin, sqrt, cavlc, ctrl and router, LCH still processes the whole subject graph in a single sweep (1 critical node processed with 0 non-critical one in Table II). On average on those circuits, the speed up compared to DCH is only of 1.22 (versus the 2.35 on all circuits). It supports our claim that processing subgraphs helps to speed-up the construction of choices. In such case, the positive speed up is only due to the applied logic synthesis operation, i.e. *dresyn2* for LCH versus *compress1* and *compress2* for DCH.

V. Conclusions

We present LCH, a new method to construct structural choices in a subject graph. Instead of working on the whole graph, which is costly due to the complexity of SAT solvers, LCH extracts non-overlapping subgraphs and looks only for choices within those. By doing so, on average on the EPFL benchmark, LCH is 2.35x faster than its state-of-the-art counterpart while enduring only a small sacrifice of 3% in node reduction and 2% in level reduction. Meanwhile, it stays better than performing no operation by 7% in node reduction and 11% in level reduction.

As future works, we would like to rely on machine learning to identify the most interesting subgraphs to extract, and which

TABLE I
COMPARISON OF DCH, LCH AND NO CHOICE CONSTRUCTOR (NONE) ON THE EPFL BENCHMARKS.

	Pre-mapping				Time (milliseconds)			Node (post-mapping)					Level (post-mapping)				
Circuits	i	o	node	level	DCH	LCH	DCH/LCH	none	DCH	LCH	none/LCH	DCH/LCH	none	DCH	LCH	none/LCH	DCH/LCH
adder	256	129	1020	255	138.6	82.6	**1.68**	254	273	269	0.94	**1.01**	51	51	51	**1.00**	**1.00**
bar	135	128	3336	12	447.8	231.6	**1.93**	512	512	512	**1.00**	**1.00**	4	4	4	**1.00**	**1.00**
div	128	128	57247	4372	10680	4022.3	**2.66**	22031	7326	6535.3	**3.37**	**1.12**	864	857	805.6	**1.07**	**1.06**
hyp	256	128	214335	24801	603854	31042	**19.45**	44503	46427	52031.3	0.86	0.89	4194	4173	3092.3	**1.36**	**1.35**
log2	32	32	32060	444	9034.2	4829.3	**1.87**	8008	8270	8458.3	0.95	0.98	77	67	66.3	**1.16**	**1.01**
max	512	130	2865	287	508.2	281.3	**1.81**	842	820	834.6	**1.01**	0.98	56	34	39.6	**1.41**	0.86
multiplier	128	128	27062	274	3645	3757.6	0.97	5913	6090	6072.3	0.97	**1.00**	53	53	53	**1.00**	**1.00**
sin	24	25	5416	225	1525.4	1434	**1.06**	1458	1473	1449	**1.01**	**1.02**	42	35	35	**1.20**	**1.00**
sqrt	128	64	24618	5058	4842.6	7291.6	0.66	5720	5071	4967	**1.15**	**1.02**	1033	990	1022	**1.01**	0.97
square	64	128	18486	250	2570.6	1813	**1.42**	3985	4255	4008.3	0.99	**1.06**	50	50	44.3	**1.13**	**1.13**
arbiter	256	129	11839	87	8022	1727.6	**4.64**	2722	2722	2721	**1.00**	**1.00**	18	18	18	**1.00**	**1.00**
cavlc	10	11	693	16	121	72	**1.68**	122	116	115	**1.06**	**1.01**	4	4	4	**1.00**	**1.00**
ctrl	7	26	175	10	18	16.6	**1.08**	29	29	29	**1.00**	**1.00**	2	2	2	**1.00**	**1.00**
dec	8	256	304	3	48	5	**9.60**	287	287	287	**1.00**	**1.00**	2	2	2	**1.00**	**1.00**
i2c	147	142	1357	20	190	49.3	**3.85**	365	340	355	**1.03**	0.96	4	3	4	**1.00**	0.75
int2float	11	7	260	16	39	28	**1.39**	49	47	50	0.98	0.94	3	3	3	**1.00**	**1.00**
mem_ctrl	1204	1231	47110	114	23534	2282.3	**10.31**	12096	11541	11921.6	**1.01**	0.97	25	20	24	**1.04**	0.83
priority	128	8	978	250	182	83.3	**2.18**	219	183	228.6	0.96	0.80	31	27	29	**1.07**	0.93
router	60	30	284	54	62	37.6	**1.65**	91	92	87	**1.05**	**1.06**	11	6	6	**1.83**	**1.00**
voter	1001	1	13758	70	6084	1984	**3.07**	2818	1505	2169.6	**1.30**	0.69	17	12	14	**1.21**	0.86
Geo Mean							**2.35**				**1.07**	**0.97**				**1.11**	**0.98**

TABLE II
NUMBER OF NODES PROCESSED AND NUMBER OF GENERATED CHOICES BY DCH AND LCH ON THE EPFL BENCHMARK.

Circuits	AIG Nodes	LCH proc.	Crit. nodes	Non-crit. nodes	DCH choices	LCH choices
adder	1020	1014	2	0	255	120
bar	3336	3336	1	0	516	256
div	57247	31722	27	2	1204	5981
hyp	214335	169367	94	897	22244	18197
log2	32060	30787	7	39	5152	4692
max	2865	2655	11	3	750	479
multiplier	27062	27061	1	2	4611	5090
sin	5416	5416	1	0	834	581
sqrt	24618	24618	1	0	4086	3136
square	18486	17814	27	16	4178	2210
arbiter	11839	11763	67	0	259	73
cavlc	693	693	1	0	212	124
ctrl	175	174	1	0	57	39
dec	304	0	0	0	0	0
i2c	1357	365	1	5	356	70
int2float	260	260	1	1	69	39
mem_ctrl	47110	18133	21	141	10721	1667
priority	978	565	1	4	254	132
router	284	257	1	0	25	15
voter	13758	9482	70	22	3228	1407

operations to perform on it. LCH could also be parallelised over multiple threads to achieve even higher speedup thanks to our non-overlapping strategy.

REFERENCES

[1] J. Cong and S. Xu, "Technology Mapping for FPGAs with Embedded Memory Blocks" in Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, pp 179–188, 1998.

[2] A. H. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for FPGA technology mapping," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 11, pp. 1319-1332, Nov. 1994

[3] K. Keutzer, "DAGON: Technology Binding and Local optimisation by DAG Matching" in 24th ACM/IEEE Design Automation Conference, pp 341-347, 1987

[4] K. Chaudhary, and M. Pedram, "A near optimal algorithm for technology mapping minimizing area under delay constraints" in Proceedings of 29th ACM/IEEE Design Automation Conference, pp 492-498, 1992.

[5] G. Liu, and Z. Zhang, "A parallelized iterative improvement approach to area optimization for lut-based technology mapping" vol. 17, pp 147–156 (Association for Computing Machinery, New York, NY, USA, 2017)

[6] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang and T. Kam, "Reducing structural bias in technology mapping," ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.

[7] E. Lehman, Y. Watanabe, J. Grodstein and H. Harkness, "Logic decomposition during technology mapping" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 8, 1997.

[8] A. Mishchenko, R. Brayton, and S. Jang, "Global Delay optimisation Using Structural Choices" in Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp 181-184, 2010.

[9] A. Mishchenko, S. Chatterjee, R. Jiang, and R. Brayton, ”FRAIGs: A Unifying Representation for Logic Synthesis and Verification”, 2005

[10] S. Palnitkar, ”Verilog HDL: a guide to digital design and synthesis”. Prentice-Hall, Inc., USA. 1996.

[11] A. Mishchenko and R. Brayton, “Scalable logic synthesis using a simple circuit structure,” in Int. Workshop on Logic and Synthesis, 2006, pp. 15–22.

[12] A. Mishchenko, S. Chatterjee, and R. Brayton. 2006. ”Improvements to technology mapping for LUT-based FPGAs”, In Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays (FPGA ’06). Association for Computing Machinery, New York, NY, USA, 41–49.

[13] A. Mishchenko, S. Cho, S. Chatterjee and R. Brayton, ”Combinational and sequential mapping with priority cuts,” 2007 IEEE/ACM International Conference on Computer-Aided Design, 2007, pp. 354-361

[14] M. Ganay and A. Kuehlmann, “On-the-fly compression of logical circuits” in Proceedings of International Workshop on Logic and Synthesis (IWLS), 2000.

[15] N. Li and E. Dubrova, ”AIG rewriting using 5-input cuts,” 2011 IEEE 29th International Conference on Computer Design (ICCD), 2011.

[16] A. Mishchenko, S. Chatterjee and R. Brayton, ”DAG-aware AIG rewriting: a fresh look at combinational logic synthesis,” 2006 43rd ACM/IEEE Design Automation Conference, 2006.

[17] E. Testa et al., ”Extending Boolean Methods for Scalable Logic Synthesis,” in IEEE Access, vol. 8, pp. 226828-226844, 2020.

[18] A. Mishchenko, R. Brayton, S. Jang and V. Kravets, ”Delay optimization using SOP balancing,” 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2011.

[19] W. Yang, L. Wang and A. Mishchenko, ”Lazy man’s logic synthesis,” 2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2012, pp. 597-604.

[20] R. Brayton and A. Mishchenko. ”ABC: An Academic Industrial Strength Verification Tool”. In Computer Aided Verification, Tayssir Touili, Byron Cook, and Paul Jackson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 24–40. 2010.

[21] J. Cong and Y. Ding, ”FlowMap: an optimal technology mapping algorithm for delay optimisation in lookup-table based FPGA designs,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 1, pp. 1-12, Jan. 1994

[22] B. Schmitt, A. Mishchenko, and R. Brayton, ”SAT-based area recovery in structural technology mapping,” 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), 2018

[23] J. Kocnová and Z. Vasicek, ”Resynthesis of logic circuits using machine learning and reconvergent paths,” 2021 24th Euromicro Conference on Digital System Design (DSD), 2021

[24] L. Amarú et al., ”SAT-Sweeping Enhanced for Logic Synthesis,” 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020

[25] H. -T. Zhang, J. -H. R. Jiang, L. Amarú, A. Mishchenko and R. Brayton, ”Deep Integration of Circuit Simulator and SAT Solver,” 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021

[26] H. -T. Zhang, J. -H. R. Jiang and A. Mishchenko, ”A Circuit-Based SAT Solver for Logic Synthesis,” 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2021

[27] L. Amarú, P. Gaillardon, and G. De Micheli. ”The EPFL combinational benchmark suite.” Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS). No. CONF. 2015.

[28] A. Cowen-Rivers, et al. ”HEBO: Pushing The Limits of Sample-Efficient Hyper-parameter Optimisation.” Journal of Artificial Intelligence Research, vol. 74, 2022.