Customizable On-the-fly Design Space Exploration for Logic Optimization of Emerging Technologies

Siang-Yun Lee
LSI, EPFL
Lausanne, Switzerland

Heinz Riener
Cadence Design Systems
Munich, Germany

Giovanni De Micheli
LSI, EPFL
Lausanne, Switzerland

Abstract

Many problems in logic synthesis are intractable because of the large number of functionally-equivalent, but structurally different ways to represent Boolean logic. Various Boolean algebraic theorems, network manipulation operations, exact algorithms as well as scalable heuristics have been developed, providing a widely-spanned collection of transformation steps connecting functionally-equivalent designs in the feasible space. The problem of design space exploration is to find a "good" sequence of applying these transformations, which leads to a desired optimum in terms of the given cost metrics. In this work, we investigate on-the-fly design space exploration methods based on random walks in the design space of available transformations. The best optimization sequence is not predefined, but rather discovered on the fly. Our implementation is customizable both for the cost evaluation metric and for the portfolio of transformation steps, which is especially important for emerging technologies having unconventional cost metrics and dedicated optimization algorithms. We present new best results in majority-inverter graph (MIG) and adiabatic quantum-flux parametron (AQFP) synthesis problems, improving over state of the art by 7% and 21%, respectively.

Keywords: Logic synthesis, combinational circuits, design space exploration, Boolean optimization

1 Introduction

Logic synthesis plays a central role in all *Electronic Design Automaton* (EDA) tools with the goal of translating a functional specification of a design into logic gates while meeting area, delay, and power quality goals. In modern tool flows, this translation process from a functional to a structural description is streamlined into a sequence of efficient, optimizing translation steps. At the technology-independent logic level, researchers propose the usage of simple logic representations [5, 20] devoid of timing and power information that are subsequently mapped [6, 23] into a technology-dependent netlist of standard cells.

The general case of the Boolean optimization problem is intractable, such that academic as well as industrial tools rely on well-tuned heuristics. Boolean optimization algorithms such as rewriting, factoring, and resubstitution [13, 19] have been revisited several times and have been improved in scalability and achievable optimization quality. Combining the individual algorithms into an efficient Boolean optimization flow, however, is rarely addressed and requires careful parameter tuning.

As a remedy, recent research proposals suggest data-driven *Artificial Intelligence* (AI) to guide logic synthesis flows and improve overall *Quality-of-Results* (QoR). An intelligent agent powered by AI could be capable of smarter decision-making by controlling when to run and stop logic optimization while considering trade-offs and conflicting QoR goals [16, 18]. Modern AI technology, however, has its own challenges: computational demands are often extraordinarily high, large amounts of training data are required, aggressive learning policies may result in biased and unexplainable decision-making, sophisticated training, and learning approaches require AI experts to design, tune, and maintain.

Moreover, with the development of beyond-CMOS emerging technologies, unconventional circuit properties, design constraints, and cost functions need to be considered in design automation. For example, *Spin Torque Majority Gate* (STMG) [17] circuits are based on majority gates and inverters are expensive, thus *majority-inverter graph* (MIG) [2] instead of *AND-inverter graph* (AIG) is a better logic network abstraction. *Adiabatic Quantum-Flux Parametron* (AQFP) [22] is also based on majority gates, and it imposes additional path-balancing and fanout-branching constraints. *Field-coupled Nanocomputing* (FCN) [3] is a family of nanotechnologies whose physical design requires the circuit to be planarized, in addition to path-balancing and fanout-branching. Although these constraints may be dealt with after technology mapping, research has shown that tailored logic optimization algorithms considering specialized cost metrics early on yield better QoR. However, carefully-tuned optimization flows for individual technologies are even more rarely researched, as such work would require experts in both the technology and logic synthesis (and AI).

In this paper, we propose a simpler design space exploration approach that takes a combinational gate-level circuit represented as input, evaluates its characteristics, and makes decisions about what optimizing transformations to apply as it proceeds. Our goal is to provide an easily-adaptable solution, customizable for various applications, when the best-achievable QoR is of interest and higher runtime is acceptable. Restart and bailout strategies are used in the exploration procedure as a mechanism to retry if a logic minimum has been reached and to terminate optimization early if QoR

deviates too much from a desired quality goal. We have implemented this decision-making process in `mockturtle` [20], a state-of-the-art logic synthesis package, and evaluated the capabilities of design space exploration with optimization problems for emerging technologies. For MIG size optimization, our new best results are 7% better than existing works. For the synthesis of AQFP circuits, we achieve 21% improvement over the state-of-the-art approach based on Bayesian optimization [9].

2 Background

2.1 Logic Optimization

Logic optimization is the problem of optimizing a logic representation to minimize multiple given (and conflicting) cost metrics. In this work, we focus on optimization of logic networks, which model digital combinational circuits. A logic network is a directed acyclic graph whose nodes model logic gates and edges model wires. Logic optimization algorithms use simple, homogeneous logic network representations [4], such as *And-Inverter Graphs* (AIGs) [11] being composed of two-fanin AND nodes and inverters and *Majority-Inverter Graphs* (MIG) [2] composed of three-fanin MAJ nodes and inverters.

2.2 Design Space Exploration

Logic optimization flows are fixed sequences of optimizing transformations. While many research works focus on improving performance and quality of individual transformations, complete optimization flows are rarely proposed. The problem of finding a sequence of optimizing transformations that achieves best results for a given benchmark suite is only recently investigated using techniques from machine learning (ML) [16, 18, 26] and Bayesian optimization [10]. These works arrange existing technology-independent optimizing transformations to reduce area and delay of the final netlist as much as possible, where each optimizing transformation maintains a local view on the logic, e.g., in the form of sliding windows, and implements a well-known scalable logic optimization. Alternatively, algorithms based on global optimization principles such as simulated annealing [15] and evolutionary algorithms [7, 8] achieve better logic compaction but are due to their high-performance requirements rarely considered in practice.

In this paper, simple, on-the-fly methods for design space exploration are investigated to find (or tune) a logic synthesis flow without requiring a complex search-based or AI-driven optimization framework. Despite the simplicity, we show that on-the-fly design space exploration can compete with flows designed by human experts and present best results in the context of AQFP synthesis and MIG size optimization.

Algorithm 1: On-the-fly design space exploration

Input: Original network N_0
Output: Optimized network N_{best}
Custom functions: `cost, decompress, compress`
Parameters: *num_restarts, max_steps, max_no_impr, timeout, init_seed*

```
1  N_best ← N_0.copy()
2  R_1 ← random_engine(init_seed)
3  for restart = 1 upto num_restarts do
4      N_best_inner ← N_0.copy()
5      N_curr ← N_0.copy()
6      R_2 ← random_engine(R_1.rand())
7      elapsed_time ← 0; start_timer(elapsed_time)
8      for step = 1 upto max_steps do
9          decompress(N_curr, R_2.rand())
10         compress(N_curr, R_2.rand())
11         if cost(N_curr) < cost(N_best_inner) then
12             N_best_inner ← N_curr.copy()
13             last_impr ← step
14         else if step − last_impr ≥ max_no_impr then
15             break
16         else if elapsed_time ≥ timeout then
17             break
18     if cost(N_best_inner) < cost(N_best) then
19         N_best ← N_best_inner.copy()
20 return N_best
```

3 On-the-fly Design Space Exploration

3.1 Overview

An overview of the on-the-fly design space exploration algorithm is outlined in Algorithm 1. Like most logic network optimization algorithms, it takes an original network as input and outputs an optimized network. Additionally, there are three custom functions a user should specify: cost evaluation, decompressing and compressing scripts, which will be further described in Sections 3.2 and 3.5.

There are an outer loop (lines 3-19) and an inner loop (lines 8-17) in Algorithm 1. In the following, we call an iteration of the outer loop a *restart* and an iteration of the inner loop a *step*. Furthermore, an execution of `decompress` (line 9) or `compress` (line 10) is called a *script*, which may contain one or more algorithms or transformations.

In each restart, the network is restored to the original one, and a new random engine seeded with a different seed is generated (line 6). The number of restarts is defined by the user (parameter *num_restarts*). The best network having the smallest cost in all restarts is recorded and eventually output by the algorithm (lines 18-20). Each restart has its own timer to upper-bound the runtime (line 7).

Each step consists of a call to a decompressing script followed by a call to a compressing script, which are both randomized. After these transformations are done, the network cost is evaluated. The current network is recorded if its cost is

the best seen among the steps executed so far in the current restart (lines 11-12). The inner loop breaks if there have been *max_no_impr* steps executed without seeing better network (lines 14-15), or if the timeout limit has reached (lines 16-17).

In the remaining of this section, we explain why we believe such algorithmic design helps achieving better design space exploration.

3.2 Escaping Local Optimum

Although a user of our algorithm has the freedom to define any set of decompressing and compressing scripts, we encourage them to classify possible transformation algorithms into two categories and have good candidates in both. A decompressing script should be a script that dramatically restructures the network and likely increases its size and depth. A prominent example of a decompressing script is LUT mapping followed by naive resynthesis to convert back into the original representation (e.g. AIG or MIG). Another example, when the representation is an MIG, is randomly breaking each majority gate into four using the relation

$$\begin{aligned} M(a,b,c) &= ab + c(a+b) \\ &= M(M(a,b,0),M(c,M(a,b,1),0),1). \end{aligned} \quad (1)$$

The purpose of decompressing is to create possibility of escaping from local optima. Imagine if the design space of all feasible networks is projected to the x axis and the y axis is the cost of each network. Such curve is very likely not convex and many valleys of local minima exist. When we are stuck at a local minimum, decompressing scripts help us to climb up the hills and potentially reaching a better local minimum afterwards.

In contrast, a compressing script is a sequence of algorithms which attempts to optimize for the given cost metric. Examples of compressing scripts include well-known logic optimization algorithms such as rewriting, balancing, refactoring, resubstitution, graph remapping, etc. The aim of compressing scripts is to converge to a local minimum. By interleaving decompressing and compressing scripts, our algorithm may explore different local optima in the design space, instead of being trapped in the nearest local optima when only applying one optimization algorithm.

3.3 Stretching Out in the Design Space

Consider the original network N_0 and a certain optimized network N_{best} to be reached, they may be far away in the design space and a long sequence of transformations is required to get from N_0 to N_{best}. Thus, our design space exploration strategy aims at stretching far out and really performing long transformation sequences. The key to such aim is that in the inner loop, even if the cost is getting much worse, there is no mechanism to backtrack to the previous best result or to retrieve the original network. A design space exploration strategy which tries many different combinations of transformation sequences but frequently backtracks would explore the design space more densely near the original network, but less likely to reach out to further points.

3.4 On-the-fly Exploration

Being able to try long sequences of transformations is not enough. The next big question is: What kind of sequence leads to better result? Although machine-learning-based research and human expert experiences give some insights, we argue that the answer is different for different benchmarks and different cost metrics. Instead of pre-defining particular sequences, our algorithm simply performs random walks. The purpose of the outer loop is to mitigate the possibility of a "bad" random seed leading to unsatisfactory result, and to increase the chance of meeting at least one "good" random sequence in all restarts. We call such strategy an *on-the-fly* exploration because we do not know the best transformation sequence in advance, but discover it on the fly during exploration.

3.5 Customization

Aiming at applications to emerging technologies with diverse logic representations, dedicated algorithms, and cost evaluation metrics, our algorithm is customizable in these aspects.

- Logic representation: As long as the transformation scripts and cost evaluation function are compatible, there is no limitation on the data structure of N_0. Although this paper focuses mainly on network optimization, it is also possible to use other logic representations such as two-level forms.
- Decompressing and compressing scripts: To set up the algorithm, the user must provide a nonempty set of decompressing scripts and a nonempty set of compressing scripts. When the functions `decompress` and `compress` (line 9 and 10 in Algorithm 1) are called, one of the scripts in the respective set is randomly chosen. The user may also define the probability of each script being chosen, preferring some scripts over the others. Moreover, how randomness is involved in the scripts is also customizable. For example, a user may define that the cut size to be used in resubstitution is randomly chosen within a range.
- Cost evaluation: Most importantly, the cost evaluation function is customized. Such function should take a network as input and output a number. It should not modify the network, but it may execute complicated algorithms to compute the cost.

Besides the custom functions, there are also some parameters users may set according to their needs.

- *num_restarts*: This parameter defines how many different transformation sequences, or exploration paths, will be tried randomly. We will experiment on the impact of this parameter in Section 4.4.

- *max_steps, max_no_impr, timeout*: These parameters define the optimization effort of each restart. Particularly, *max_no_impr* defines how many steps without seeing any improvement in the cost the algorithm will tolerate before bailing out from the current exploration path, and *timeout* defines the runtime budget.
- *init_seed* is the user-specified initial random seed used to generate different seeds to be used in each restart. This parameter is only used to ensure deterministic and reproducible results of the algorithm. When *num_restarts* is sufficiently large (Section 4.4), different *init_seed* should give similar results and tuning of this parameter should not be needed.

4 Experimental Results

The proposed design space exploration algorithm is implemented in the C++ logic synthesis library `mockturtle`[1] [20, 21]. Two benchmark suites are used for experiments to match the respective state-of-the-art works in comparison. The MCNC benchmarks [25] are used in Section 4.2 for AQFP synthesis and the EPFL benchmarks [1] are used in Section 4.1 for MIG size optimization.

4.1 Application to MIG Optimization

Table 1 compares a state-of-the-art MIG restructuring algorithm, graph remapping [23] (Map), the current best MIG size results seen in the literature produced by an optimization flow [12] (Flow), and the new best results achieved by our design space exploration (DSE). The MIG sizes (number of gates) are listed for all of the three as the main comparison, and the MIG depth is additionally listed in DSE for reference. The benchmark suite is divided into arithmetic circuits (upper half) and control circuits (lower half), and the sum of arithmetic benchmarks as well as all benchmarks are listed separately. Data of the control circuits for Map were omitted in the table because they were not presented in [23].

From Table 1, we also observe the improvements made by extending from a single algorithm, to a fixed flow, and finally to an exploration of a portfolio of different flows. Overall, our new best result improves over state of the art by 7.2%.

Table 1. Comparison on MIG size against previous works.

	Map [23]	Flow [12]	DSE [Ours]	
Bench.	Size	Size	Size	Depth
adder	384	384	384	129
bar	2588	2588	2112	14
div	36858	12532	12480	2267
hyp	137048	124177	121674	8731
log2	24295	23109	22996	187
max	2171	2210	1968	217
multiplier	19299	18440	18437	138
sin	4196	3967	3963	116
sqrt	17355	12423	12384	2193
square	11924	9498	8732	127
Total (arith.)	256118	209328	205130	14119
arbiter	-	6719	3907	23
cavlc	-	533	381	16
ctrl	-	79	63	8
dec	-	304	304	3
i2c	-	932	652	22
int2float	-	181	121	14
mem_ctrl	-	34777	24029	44
priority	-	431	366	34
router	-	151	105	11
voter	-	4561	4351	31
Total (all)	-	257996	239409	14325

4.2 Application to AQFP Synthesis

Adiabatic quantum-flux parametron (AQFP) is an emerging superconducting digital electronic technology featuring ultra-low energy consumption [22]. AQFP circuits are based on the majority logic and imposes special design constraints on path balancing and fanout branching. To address these, MIGs are often used as the logic representation, and buffers and splitters are inserted to fulfill the constraints. Due to the complex interplay between equivalent logic structures, path balancing and fanout branching, restructuring and optimization of AQFP circuits is a difficult problem and classical logic synthesis algorithms cannot be directly applied. Two orthogonal approaches have been proposed: conventional MIG synthesis followed by buffer and splitter insertion and optimization [24], and dedicated synthesis algorithms considering AQFP cost in logic restructuring [14]. The cost of an AQFP circuit is commonly evaluated with the Josephson junction (JJ) count (#JJs $= 6 \cdot$ #MAJs $+ 2 \cdot$ #buffers) and the circuit depth (length of critical path counting both gates and buffers).

Table 2 presents our new best results for the AQFP synthesis problem using the proposed design space exploration algorithm and compares to the state-of-the-art approach based on Bayesian optimization [9]. The optimization goal is set to minimizing the energy-delay product (product of JJ count and circuit depth) to match with the compared work (SoTA). The JJ count (#JJs), circuit depth (Depth) and energy-delay product (EDP) are listed, as well as the difference in EDP (Δ). On average, our design space exploration improves EDP by 21.4% and JJ count by 31.6%.

4.3 Design Space Exploration

We take the benchmark "5xp1" from the AQFP synthesis experiment in Section 4.2 and plot the processes of three restarts in Figure 1 as an example illustration of design space

[1] Available: https://github.com/lsils/mockturtle

Table 2. Comparison on JJ count, depth, and energy-delay product against state of the art AQFP synthesis.

	SoTA [9]			Ours			
Bench.	#JJs	Depth	EDP	#JJs	Depth	EDP	Δ
5xp1	726	10	7260	418	14	5852	-19%
c1908	5108	34	173672	4620	37	170940	-2%
c432	3098	34	105332	2370	38	90060	-14%
c5315	16410	30	492300	15648	30	469440	-5%
c880	3876	23	89148	3836	28	107408	+20%
chkn	3500	15	52500	2142	16	34272	-35%
count	1400	12	16800	2014	25	50350	+200%
dist	3536	14	49504	1910	21	40110	-19%
in5	3370	14	47180	1684	17	28628	-39%
in6	2884	11	31724	1862	14	26068	-18%
k2	14748	22	324456	8660	25	216500	-33%
m3	2680	12	32160	1602	15	24030	-25%
max512	4812	16	76992	2904	18	52272	-32%
misex3	11272	20	225440	2826	20	56520	-75%
mlp4	2976	14	41664	1650	17	28050	-33%
prom2	22326	20	446520	16094	21	337974	-24%
sqr6	916	10	9160	700	11	7700	-16%
x1dn	1208	11	13288	766	13	9958	-25%
Total	104846	322	2235100	71706	380	1756132	-21%

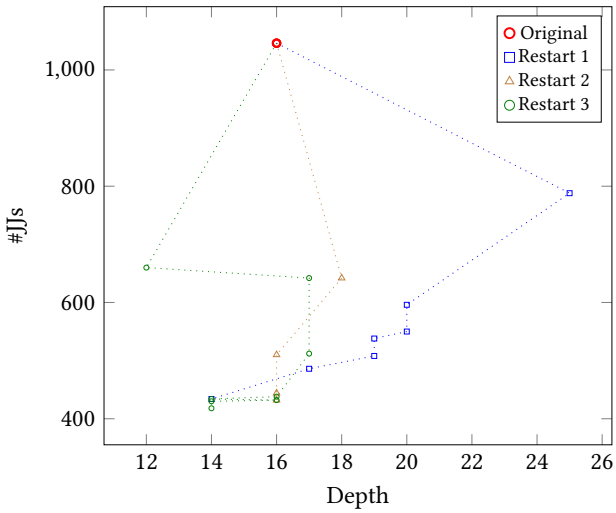

Figure 1. Three different paths in the design space taken by three restarts.

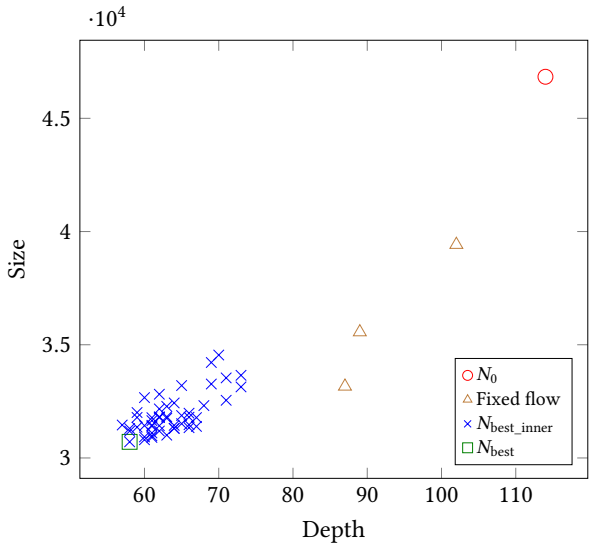

Figure 2. Local optima found by 50 restarts (×) compared to a fixed flow (△).

exploration. The optimization goal is set to minimizing JJ count (y-axis), and the circuit depth is used as the x-axis of the plot to help distinguish different networks seen in the process. Only the networks causing an update to $N_{\text{best\_inner}}$ are recorded. We observe the different paths taken by the design space exploration algorithm.

4.4 Importance of Random Restarts

To investigate the influence of different random seeds used in each restart, we plot the best-seen network in 50 restarts in the same run. The benchmark "mem_ctrl" from the EPFL benchmark suite is used and optimized for MIG size. In Figure 2, the y-axis is MIG size (optimization goal) and the

x-axis is MIG depth (a second network trait). Each blue cross is a local optimum $N_{\text{best\_inner}}$ recorded after 50 steps of transformation without improvement or when the inner loop times out, and the green square marks the best among the 50 restarts. The red circle is the initial network N_0, and the brown crosses are the results of fixed, predefined flows designed by human experts.

We observe from this experiment that there really exist many different local minima in the design space. Some of them are worse in both metrics, and some of them form a portion of the pareto curve. As the algorithm is a random process, the order of encountering them is random. If *num_restarts* was set smaller, the chance of getting the same best local optimum is reduced.

However, there are not infinite local minima and increasing *num_restarts* indefinitely may not always help finding a better optimum. We have observed that for some benchmarks and settings, many restarts fall into the same few local minima.

5 Conclusion and Discussions

This work presents an on-the-fly design space exploration algorithm which emphasizes long transformation sequences and restarts with different random decisions. The implementation is customizable for unconventional cost functions often seen in emerging technologies, as well as dedicated, customized optimization scripts. With the proposed design space exploration, we are able to improve over state-of-the-art QoRs on MIG and AQFP optimization problems.

We study the different trajectories of design space exploration and experimentally show that there may be many different local optima reachable by different flows found by the design space exploration algorithm. We argue that there does not exist a fixed universally-good flow that works well for all benchmarks, so that the search of the best flow shall be done on the fly. As future work, we would like to experimentally demonstrate this claim by applying the best flow found for one benchmark on another benchmark.

Randomized decision is key to the proposed algorithm because it is the premise of forming different flows and taking different trajectories leading to different local optima. The algorithm would not work if there is only one unrandomized script provided. However, it remains an open question how many different scripts do we need. We conjecture that the more randomization involved, the wider distribution of local optima we will get in a plot similar to Figure 2. In other words, better optima would become reachable, but there will be more worse optima as well. Further experiments are required to answer this question.

As another future research direction, we would also like to explore the possibility of learning from the best flows found by random exploration for various benchmarks. Will there be a trend or similarities among these flows? For example, perhaps a certain transformation script is particularly important and is involved in all flows.

Acknowledgments

This work was supported in part by the SNF grant "Supercool: Design methods and tools for superconducting electronics", 200021_1920981.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Proceedings of IWLS*.

[2] Luca Amaru, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. Majority-inverter graph: A new paradigm for logic optimization. *IEEE Trans. on CAD* 35, 5 (2015), 806–819.

[3] Neal G. Anderson and Sanjukta Bhanja (Eds.). 2014. *Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives*. Lecture Notes in Computer Science, Vol. 8280. Springer.

[4] Robert K. Brayton, Gary D. Hachtel, and Alberto L. Sangiovanni-Vincentelli. 1990. Multilevel logic synthesis. *Proc. IEEE* 78, 2 (1990), 264–300. https://doi.org/10.1109/5.52213

[5] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification, 22nd International Conference, CAV 2010*. 24–40.

[6] Jason Cong and Yuzheng Ding. 1994. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 13, 1 (1994), 1–12. https://doi.org/10.1109/43.273754

[7] Petra Färm, Elena Dubrova, and Andreas Kuehlmann. 2011. Integrated logic synthesis using simulated annealing. In *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*. 407–410.

[8] Petr Fiser, Jan Schmidt, Zdenek Vasícek, and Lukás Sekanina. 2010. On logic synthesis of conventionally hard to synthesize circuits using genetic programming. In *13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2010, Vienna, Austria, April 14-16, 2010*, Elena Gramatová, Zdenek Kotásek, Andreas Steininger, Heinrich Theodor Vierhaus, and Horst Zimmermann (Eds.). IEEE Computer Society, 346–351. https://doi.org/10.1109/DDECS.2010.5491755

[9] Rongliang Fu, Junying Huang, Mengmeng Wang, Yoshikawa Nobuyuki, Bei Yu, Tsung-Yi Ho, and Olivia Chen. 2023. BOMIG: A Majority Logic Synthesis Framework for AQFP Logic. In *DATE'23*.

[10] Antoine Grosnit, Cédric Malherbe, Rasul Tutunov, Xingchen Wan, Jun Wang, and Haitham Bou-Ammar. 2022. BOiLS: Bayesian Optimisation for Logic Synthesis. In *DATE 2022*, Cristiana Bolchini, Ingrid Verbauwhede, and Ioana Vatajelu (Eds.). IEEE, 1193–1196.

[11] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. 2002. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 21, 12 (2002), 1377–1394. https://doi.org/10.1109/TCAD.2002.804386

[12] Siang-Yun Lee and Giovanni De Micheli. 2023. Heuristic Logic Resynthesis Algorithms at the Core of Peephole Optimization. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* (2023).

[13] Siang-Yun Lee, Heinz Riener, Alan Mishchenko, Robert K. Brayton, and Giovanni De Micheli. 2022. A Simulation-Guided Paradigm for Logic Synthesis and Verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41, 8 (2022), 2573–2586.

[14] Dewmini Sudara Marakkalage, Heinz Riener, and Giovanni De Micheli. 2021. Optimizing Adiabatic Quantum-Flux-Parametron (AQFP) Circuits using an Exact Database. In *IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2021, AB, Canada, November 8-10, 2021*. IEEE, 1–6. https://doi.org/10.1109/NANOARCH53687.2021.9642241

[15] Julian F Miller, Dominic Job, and Vesselin K Vassilev. 2000. Principles in the evolutionary design of digital circuits—Part I. *Genetic programming and evolvable machines* 1, 1 (2000), 7–35.

[16] Walter Lau Neto, Yingjie Li, Pierre-Emmanuel Gaillardon, and Cunxi Yu. 2022. FlowTune: End-to-end Automatic Logic Optimization Exploration via Domain-specific Multi-armed Bandit. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2022).

[17] Dmitri E Nikonov, George I Bourianoff, and Tahir Ghani. 2011. Proposal of a spin torque majority gate logic. *IEEE Electron Device Letters* 32, 8 (2011), 1128–1130.

[18] Yasasvi V. Peruvemba, Shubham Rai, Kapil Ahuja, and Akash Kumar. 2021. RL-Guided Runtime-Constrained Heuristic Exploration for Logic Synthesis. In *ICCAD 2021*. IEEE, 1–9.

[19] Heinz Riener, Winston Haaswijk, Alan Mishchenko, Giovanni De Micheli, and Mathias Soeken. 2019. On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, Jürgen Teich and Franco Fummi (Eds.). IEEE, 1649–1654. https://doi.org/10.23919/DATE.2019.8715185

[20] Heinz Riener, Eleonora Testa, Winston Haaswijk, Alan Mishchenko, Luca G. Amarù, Giovanni De Micheli, and Mathias Soeken. 2019. Scalable Generic Logic Synthesis: One Approach to Rule Them All. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 70. https://doi.org/10.1145/3316781.3317905

[21] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Siang-Yun Lee, Alessandro Tempia Calvino, Dewmini Sudara Marakkalage, and Giovanni De Micheli. 2022. The EPFL Logic Synthesis Libraries. arXiv:1805.05121 http://arxiv.org/abs/1805.05121

[22] Naoki Takeuchi, Dan Ozawa, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2013. An adiabatic quantum flux parametron as an ultra-low-power logic device. *Superconductor Science and Technology* 26, 3 (2013), 035010.

[23] Alessandro Tempia Calvino, Heinz Riener, Shubham Rai, Akash Kumar, and Giovanni De Micheli. 2022. A Versatile Mapping Approach for Technology Mapping and Graph Optimization. In *ASP-DAC 2022*. 410–416.

[24] Eleonora Testa, Siang-Yun Lee, Heinz Riener, and Giovanni De Micheli. 2021. Algebraic and Boolean Optimization Methods for AQFP Superconducting Circuits. In *ASPDAC '21: 26th Asia and South Pacific Design Automation Conference, Tokyo, Japan, January 18-21, 2021*. ACM, 779–785. https://doi.org/10.1145/3394885.3431606

[25] Saeyang Yang. 1991. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC).

[26] Cunxi Yu, Houping Xiao, and Giovanni De Micheli. 2018. Developing Synthesis Flows without Human Knowledge. *Design Automation Conference (DAC'18)* (June 2018).