HAL open science

A Design Space Exploration Framework for Memristor-Based Crossbar Architecture

Mario Barbareschi, Alberto Bosio, Ian O'Connor, Petr Fiser, Marcello Traiola

To cite this version:

Mario Barbareschi, Alberto Bosio, Ian O'Connor, Petr Fiser, Marcello Traiola. A Design Space Exploration Framework for Memristor-Based Crossbar Architecture. DDECS 2022 - 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, Apr 2022, Prague, Czech Republic. pp.38-43, 10.1109/DDECS54261.2022.9770145 . hal-03888009

HAL Id: hal-03888009

https://inria.hal.science/hal-03888009v1

Submitted on 3 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

A Design Space Exploration Framework for Memristor-Based Crossbar Architecture

Mario Barbareschi[1],Alberto Bosio[2], Ian O'Connor[2], Petr Fišer[3], and Marcello Traiola[4]
[1]*Department of Electrical Engineering and Information Technology, University of Naples Federico II, Naples, Italy*
[2]*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France*
[3]*Czech Technical University in Prague, Czech Republic*
[4]*University of Rennes, Inria, CNRS, IRISA, UMR6074*
Email: mario.barbareschi@unina.it, alberto.bosio@ec-lyon.fr, ian.oconnor@ec-lyon.fr
fiserp@fit.cvut.cz, marcello.traiola@inria.fr

***Abstract*—In the literature, there are few studies describing how to implement Boolean logic functions as a memristor-based crossbar architecture and some solutions have been actually proposed targeting back-end synthesis. However, there is a lack of methodologies and tools for the synthesis automation. The main goal of this paper is to perform a Design Space Exploration (DSE) in order to analyze and compare the impact of the most used optimization algorithms on a memristor-based crossbar architecture. The results carried out on 102 circuits lead us to identify the best optimization approach, in terms of area/energy/delay. The presented results can also be considered as a reference (benchmarking) for comparing future work.**

Index terms - Emerging Technologies, Memristor crossbar, Design Space Exploration, Boolean Functions, Logic Synthesis

I. Introduction

Today's computing devices are based on the CMOS technology, that is subject to the famous Moore's Law [1]. Despite the advantages of the technology shrinking, we are facing the physical limits of CMOS. Among the multiple challenges arising from technology nodes lower than 10 nm, we can highlight the high leakage current (i.e., high static power consumption), reduced performance gain, reduced reliability, complex manufacturing process leading to low yield and complex testing process, and extremely costly masks [2].

Several emerging technologies are under investigation, among them the memristor is a promising one [3]. The memristor is a non-volatile device able to act as both storage and information processing unit that presents many advantages: CMOS process compatibility, lower cost, zero standby power, nanosecond switching speed, great scalability, high density, and non-volatile capability [4], [5]. Thanks to its nature (i.e., computational as well as storage element), the memristor is used in different kinds of applications, such as neuromorphic systems [6], non-volatile memories [7], or computing architecture for data-intensive applications [8].

A fundamental component of any kind of computing architecture is the implementation of Boolean logic functions. In [9], the authors proposed a methodology for the synthesis of Boolean logic functions on a memristor crossbar. Their work showed that it is possible to implement any kind of Boolean function on a memristor crossbar. However, the experimental results have been carried out only on a couple of small circuits owing to the lack of a model with high abstraction level and also to the lack of an automated synthesis framework. In fact, they used a SPICE level model of the memristor crossbar that actually limits the possibility of dealing with realistic Boolean functions and of course the logic synthesis.

As defined in [10], the overall problem of logic synthesis is the one of finding "the best implementation" of a Boolean function. The term "best" corresponds to a trade-off between several metrics such as the area, delay, and power consumption. Today, with the rising of alternative technologies to CMOS, we are facing new challenges for logic synthesis. It is therefore mandatory to well understand the logic gates built on top of emerging technologies and identify the available opportunities. In [11], we presented an automatic tool for mapping a given Boolean function into a memristor crossbar. Thanks to this tool, it is possible to analyze two kinds of logic synthesis (i.e., 2-level and multi-level). In this paper, we extend [11] by considering additional approaches for logic synthesis and present a massive exploration phase. More in detail, the contributions can be summarized as follows:

- We analyze the impact of the different kinds of technology mapping to identify the best one for memristor based crossbars;
- We quantify the circuits implementation w.r.t. area/energy/delay and we made a comparison with a CMOS based implementation;
- The experimental results can be useful as a reference (benchmarking) for comparing future work. Results are published as open access repository [12].

The remainder of the paper is structured as following. Section II presents the state-of-the-art and provides the required background about the memristor based computation. Section III details the proposed memristor and crossbar model as well as the synthesis framework, while Section IV gives the experimental results. Finally, Section V draws the conclusions.

II. Background and state-of-the-art

In this section, we provide an overview of the state-of-the-art about memristor-based circuits and the existing automatic synthesis approaches.

A. State-of-the-art overview

Memristors are currently under investigation and used in different types of computing architectures: from analog computation mainly exploited in neuromorphic computing [13], [14], [15], up to

digital architectures where the memristors are used to implement Boolean functions.

In [16], the authors described the memristor-based IMPLY logic gate and proposed a methodology for designing it. Then, in [17], the authors presented an approach to synthesize memristor-based combinational logic circuits based on such logic. They proposed an evolutionary algorithm for obtaining a reduced number of working memristors. Also Raghuvanshi et al. considered implication-logic-based memristor circuits in [18]. Indeed, they illustrated new logic synthesis methods which aim at minimizing the number of implication gates, and thus the delay. Furthermore, in [19], the authors addressed the memristor-based stateful logic problems (i.e., sequential voltage activations, result stored by one of the inputs, additional circuit components required) proposing a memristor-only logic family, i.e., memristor-aided logic (MAGIC). The authors of [20] took into account implication-logic-based memristor circuits. They represented Boolean functions as Reduced Ordered Binary Decision Diagrams (ROBDDs) and mapped them to 2-to-1 multiplexers implemented with memristors.

In [21], the authors presented an approach for the synthesis of memristive-based logic circuits using the Majority-Inverter Graphs (MIGs). They proposed optimizations of MIGs to reduce the number of memristive and computational steps in both MAJ-based and IMP-based realizations. Xie et al. in [9] proposed a methodology that led to the synthesis of any kind of Boolean logic function on a memristor crossbar architecture. Afterwards, in [22], they proposed a generic synthesis framework to map logic circuits on memristor crossbars. In particular, it performs the synthesis of both combinational and sequential circuits. The framework takes HDL descriptions as input and generates circuits that can be simulated using electronic circuits simulators (i.e., SPICE). Furthermore, in [23] they illustrated a methodology to map large Boolean logic circuits on a memristor crossbar. Specifically, they presented efficient place-and-route solutions and several optimization layouts for area, delay, and power consumption. They are able to apply their methodology to large logic circuits and to evaluate the performance of the design.

The scope of this work is in the latter category: digital crossbar-array circuits. Solutions proposed in [9], [22], [23] are very useful for the automation of the Physical Synthesis process that appears in [24]). However, since there is no well-defined strategy for optimizing at the best a given Boolean function aiming to have the best memristor-based circuit according to design requirements, a massive and quick exploration phase is still needed. Thus, a higher level of abstraction is needed in order to speed up such process. We intend to investigate Logic Synthesis in order to have a comprehensive view of the optimization opportunities.

Next subsections describe the memristor model that we designed in our work. For this purpose, firstly we have to describe how the memristor is exploited in a crossbar architecture to perform a computation.

B. Memristor model

A memristor is a non-linear electrical component whose electrical resistance is not constant but depends on the history of the charge flowed through the device itself. Since we intend to implement a digital circuit, we refer to the memristor Voltage-Current relation depicted in Figure 1, detailed in [25], as the best solution for modeling the memristor's behavior (i.e., thanks to the ideal response to a pulse-wave). Thus, as depicted in Figure 1, the voltage applied to the memristor's terminals does not change its resistance until it crosses a threshold. In the adopted ideal model, the upper and the lower thresholds have the same absolute value.

We resort to the Snider Boolean Logic (SBL) [25] convention whereby a lower resistance (steeper curve denoted as R_{ON}) represents a logic '0' while a higher resistance (lower slope curve denoted as R_{OFF}) represents a logic '1'.

Two basic operations can be performed, defined as SET and RESET. The first one allows to program the memristor to R_{ON} and thus at logic '0', while the second one programs the memristor to R_{OFF} that corresponds to logic '1'. The Figure 1 depicts SET and RESET operations as described by Xie et al. in [9].

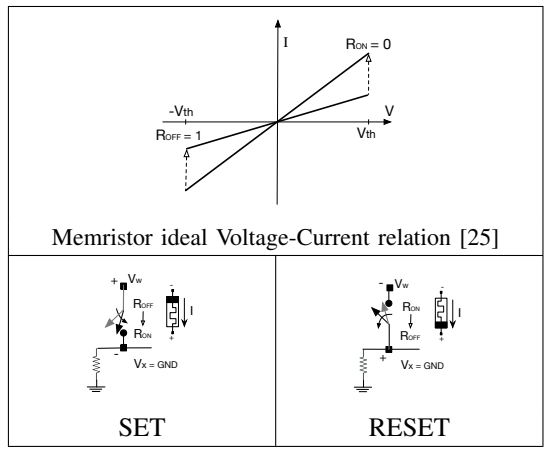

Fig. 1: Set and Reset operations [9]

C. Fast Boolean Logic Circuits

Snider proposed in [25] a design methodology to implement Boolean functions on memristor-based crossbars. This design was then improved by Xie et al. in [9]. Let us briefly recall their proposition, by referring to it as Fast Boolean Logic Circuit (***FBLC***). Let us resort to an example by considering the following Boolean function:

$$O = AB + \overline{A}B + \overline{AB} = \overline{\overline{AB} \cdot \overline{\overline{A}B} \cdot \overline{\overline{AB}}} \qquad (1)$$

The left member of the Equation 1 can be easily manipulated through transformation rules (i.e., De Morgan's laws). The obtained form (right member of the Equation 1) can be computed by using three Boolean operations: NAND, AND and NOT.

Then, as sketched in Figure 2-a, the FBLC implementation of 1 is divided in blocks:

- Input block: it stores the inputs, in the example A, B and their complements;
- Minterm block ($\overline{M}$): it configures and evaluates the minterms, in the example we have three minterms, one per row;
- AND block: it stores the results of the minterms and performs the AND between them;
- Output block: it stores the results of the AND block and performs the inversion operation to obtain the output value O in our example.

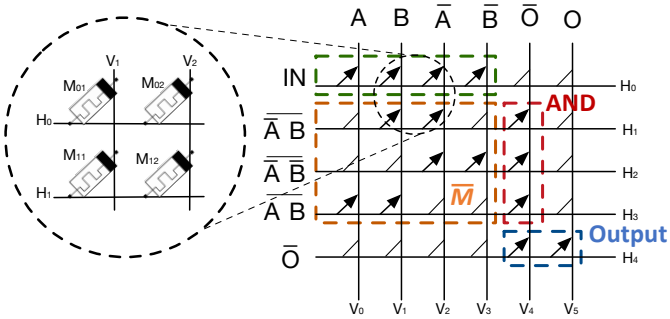

Fig. 2: FBLC example

To drive and control the different blocks a control circuitry is required. All the details can be found in [9].

III. Towards automatic crossbar synthesis

This section presents the synthesis framework and the behavior model of the memristor-based crossbar.

A. XbarGen

XbarGen [11], [12] analyzes the given input function to determine its levels. We refer to a "*level*" as a subset of the Boolean function having inputs and outputs such that:

- inputs are independent of each other; the inputs of a given level can be either the primary inputs or intermediate values;
- each output depends only on inputs (i.e, there are no intermediate nets); the outputs can be either primary outputs or intermediate values.

Secondly, the framework performs a mapping to one or more crossbars, depending on the number of levels the input function is made of. For each level, inputs, outputs, and related minterms are mapped to a FBLC crossbar, as explained in [9] and briefly reminded in the previous section. When multiple crossbars are produced, they must be connected together in series, according to what was proposed by Snider in [25]. Consequently, the latency (i.e., the computational time) of the circuit grows proportionally to the number of serially connected crossbars.

In this work, we study how different expressions of a Boolean function (i.e., logic synthesis) lead to different FBLC implementations. In turn, this will lead to to different trade-offs between performance (i.e., computational time) versus costs (i.e., area and energy consumption). The goal of the exploration is determining the best logic synthesis approach to obtain a Boolean function expression leading to the best FBLC implementation.

IV. Experiments and results

To the best of our knowledge, this is the first work comparing the effectiveness of existing logic minimization approaches for the synthesis of Boolean functions as memristor-based crossbars. In this section, we provide experimental results showing the impact of minimization approaches in terms of area/energy/delay.

We performed experiments following the flow described in Figure 4. Specifically, given a Boolean function, we employed different logic minimization (i.e., Logic Synthesis) approaches available in the state-of-the-art synthesis tool ABC [26]. In particular, we have run these processes:

- **Collapse**: Collapsing into a two-level form by using the ABC `collapse` command
- **Collapse-E**: Collapsing into a two-level form by using the ABC `collapse` command, followed by the two-level minimization by Espresso [27]
- **Collapse-Eso**: Collapsing into a two-level form by using the ABC `collapse` command, followed by the two-level minimization by Espresso [27] with the `-Dso` option (single-output minimization)
- **LUTa**: LUT mapping optimized for area by using the ABC command sequence
  `dch; if -a -K` K`; mfs -W 10; st; dc2; if -a -K` K`; mfs -W 10`
- **LUTd**: LUT mapping optimized for delay by using the ABC command sequence
  `dch; if -K` K`; mfs -W 10; st; dc2; if -K` K`; mfs -W 10`

Both LUT-mapping scripts were iterated 20-times, for different K, $2 \leq K \leq 31$. Also, note that the synthesis with $K = 2$ performed mapping using arbitrary 2-input gates. We adopted such solution since a K-LUT can be easily mapped as a crossbar by using XbarGen [11], [12]. The flow is depicted in Figure 3. Then, we employed XbarGen for obtaining statistics: occupied area, delay, and estimated energy consumption (switching activity) [28].

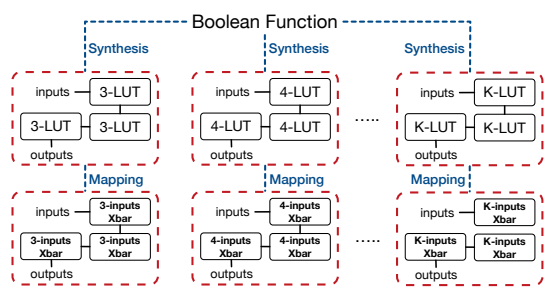

Fig. 3: Synthesis flow using K-input LUTS

We carried out experiments over 102 different combinational circuits from the LGSynth'91 [29], IWLS'93 [30], ISCAS'85 [31], ISCAS'89 [32], ITC'99 [33], EPFL [34], and IWLS 2005 [35] benchmark sets, with up to 452 inputs, 256 outputs, and up to 78 thousands literals. It is worth to mention that some circuits were obtained by extracting the combinational logic of sequential circuits.

We first detail how we compute statistics. Table I reports the three metrics that we used: Area, Delay, and Energy. **Area** depends on the number of inputs, outputs, and minterms of each level in the Boolean function. These parameters depend on the applied logic synthesis. We do not report absolute values since we do not target any specific memristor technology. Therefore, the Area is measured as "units", where each unit corresponds to the size of a memristor. The **Delay** depends on the number of steps (N_{Steps}) required to evaluate the Boolean function implementation. In the case of a single crossbar implementation, it corresponds to 7 (see Section II). The absolute time is related to the memristor technology and it is expressed as t_M: the time required to change the status of a single memristor. T_C is thus the absolute time required to obtain the output values for a single crossbar. Finally, in

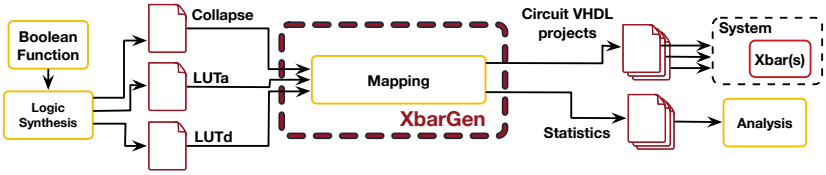

Fig. 4: Synthesis Framework Flow

TABLE I: Legend

Symbol	Description	Expressed as
p_{ij}	Whether the *i-th* minterm is in the *j-th* level (Boolean)	Given by Logic Synthesis
$N_{in}(l_j)$	Num. of input for the *j-th* level	Given by Logic Synthesis
$N_{out}(l_j)$	Num. of output for the *j-th* level	Given by Logic Synthesis
Area	Total area	$\sum_j \left\{ [2 * N_{in}(l_j) + 2 * N_{out}(l_j)] * \left[1 + \sum_i (p_{ij}) + N_{out}(l_j) \right] \right\}$
t_M	Memristor switching time	Given from technological characterization
N_{Steps}	Num. of steps of a Crossbar computation	Given by the architecture (equal to 7)
T_C	"Latency" of a single Crossbar	$t_M * N_{Steps}$
N_C	Num. of Crossbar in the circuit	Given by Logic Synthesis
Delay	Needed Cycles to complete while computation	$T_C * N_C$
Energy	Computed as the worst case of memristors switching activity	Given from XbarGen, details available in [28]

the case of several crossbars, the overall Delay time is computed as T_C multiplied by N_C. Again, the number of crossbars is provided by the logic synthesis. The last metric is the **Energy**. As for previous ones, it is related to the number of memristors used for implementing the given Boolean function. To quantify the energy consumption, we do not perform any simulations, we simply apply the methodology described in [28] allowing to estimate the average case (i.e., the average energy consumption).

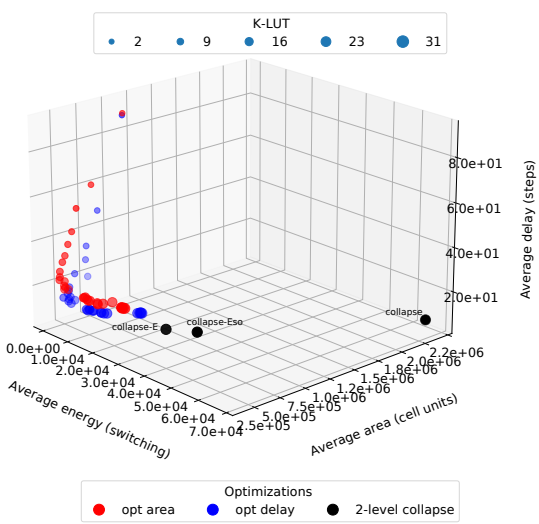

Fig. 5: Results: Area vs. Delay vs. Energy

In Figure 5, we report the 3D representation of the obtained results w.r.t. the three metrics. To better discuss the results, we present three views obtained from Figure 5 by plotting two metrics at a time. This leads to charts in Figures 6, 7, and 8. For all charts, we depicted as blue dots the solutions LUTd (i.e., optimized w.r.t. delay) and as red dots the solutions LUTa (i.e., optimized w.r.t. area). Each dot is further characterized by its K value. In the figures, larger the K value, larger the plotted point dimensions. The black dot always refers to the "Collapse" solution (i.e., the two-level synthesis). Last information about the charts is related to the fact that each dot (blue, red and black) corresponds to the average of the metrics extracted by synthesizing and mapping to the FBLC all 102 circuits (i.e., the overall number of syntheses is thus 3,366).

A. Area Vs. Delay

Figure 6 reports the trade-off between Area and Delay. First of all, it is clear from the results that the applied logic synthesis has a huge impact on the resulting circuit. Indeed, by using K-LUT mapping with $K = 11$ we obtained the best area. Increasing the K value makes the area larger while the delay drops. By collapsing the design to 2 levels, the best delay is obtained, but also the larger area. In this sense, we find the well-known area-delay trade-off characterizing the standard CMOS digital circuits. On the contrary, for K-LUT mappings with $K < 11$, both area and timing increase/decrease at the same time, thus not following the area-delay trade-off.

B. Energy Vs. Area

Figure 7 reports the relation between Energy and Area. First of all, the Collapse logic synthesis appears not to be the ideal solution. On the other hand, the LUT based synthesis with $K = 11$ provides one order of magnitude less energy consumption than the Collapse solution, with also smaller Area. For $K > 11$, both area and energy increase. This is expected, since bigger circuits usually consume more energy. On the contrary, for $K < 11$, energy consumption stays roughly constant around 10^3 while area changes in the range $10^5 - 10^6$.

C. Energy Vs. Delay

Figure 8 reports the relation between Energy and Delay. This chart is qualitatively similar to Figure 7 (i.e., for $K < 11$, energy consumption stays roughly constant and for $K > 11$, it increases while the delay decreases). Again, this was expected, since Delay

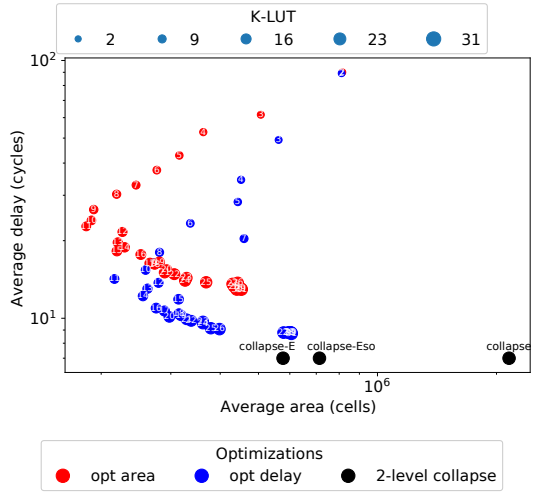

Fig. 6: Results: Area vs. Delay

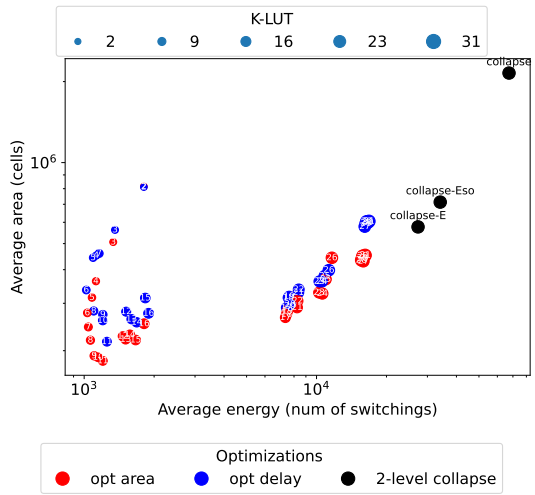

Fig. 7: Results: Area vs. Energy

depends on the Area (see Figure 6). Previous observations are thus valid in this case too.

D. Comparison with CMOS

Finallty, Figure 9 wants to give an idea of the obtained results w.r.t. the CMOS counterpart, in terms of area and area-delay product. CMOS implementations were obtained by synthesizing the circuits with a standard 45nm 2-input gate library [36]. To achieve a meaningful comparison, we needed technological parameter for the memristors: we used the following parameters, according to [37], $3nm \times 3nm$ cell-size and 1 ns switching delay. As shown in the figure, the memristor implementations outperform the CMOS in terms of area. Furthermore, also in terms of area-delay product, we observe a similar result. Finally, since we did not model the system level, we do not compare memristor and CMOS implementations w.r.t. energy consumption and delay. Indeed, both should be measured while taking into account also data movement, which strictly depends on the system level.

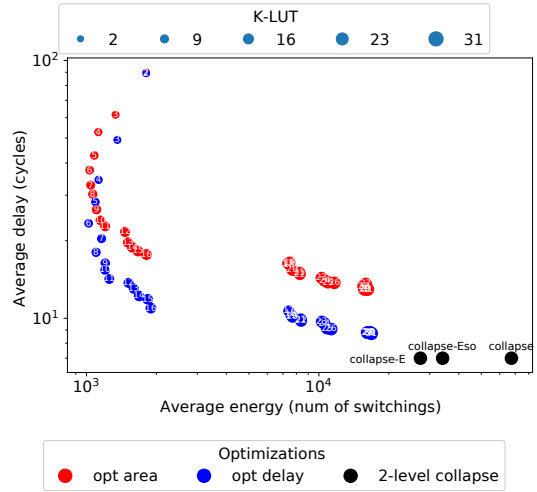

Fig. 8: Results: Energy vs. Delay

V. Conclusion

The memristor is one of the most promising technologies which are able to deal with the CMOS limitations. In particular, since the memristor is inherently able to behave also as a non-volatile device, it allows overcoming the bottleneck of the data transfer to the computational unit and back to storage elements. The research community is facing the synthesis of Boolean functions by utilizing memristor-based crossbars. In order to speed up the exploration of new memristor-based crossbar architectures, our proposition is to move the analysis to a higher level (i.e., Behavioral).

The main goal was the analysis of the well-known logic optimization techniques such as the 2-level and the multi-level optimizations. Moreover, we also identified the best technology mapping process to further improve the optimization of logic synthesis for memristor-based crossbars. To achieve these results, we presented a Synthesis framework flow to explore the optimization process targeting a memristor-based architecture (i.e., FBLC). In the first phase, we applied Logic Synthesis to obtain a two-level (Collapse) or a LUT based implementation. Then, we map the results of Logic synthesis in memristor crossbar(s) and we quantified the final implementation by using Area, Delay and Energy as metrics. The above framework has been applied to a set of 102 benchmarks leading to 3,366 different implementations.

Results prove that K-LUT optimization approach provides the best results when considering Area/Energy/Delay as metrics. Finally, we believe that all the presented results, tool and HDL model can be considered as a valid benchmark set for comparing future work. For this purpose, it is possible to download all the presented material in [12].

References

[1] (2013) ITRS 2013 report. [Online]. Available: http://www.itrs2.net/
[2] B. Hoefflinger, "Chips 2020: A guide to the future of nanoelectronics," in *The Frontiers Collection, Heidelberg*. Berlin: Springer, 2012, pp. 421–427.
[3] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, September 1971.
[4] R. Waser *et al.*, "Redox-based resistive switching memories - nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, 2009.

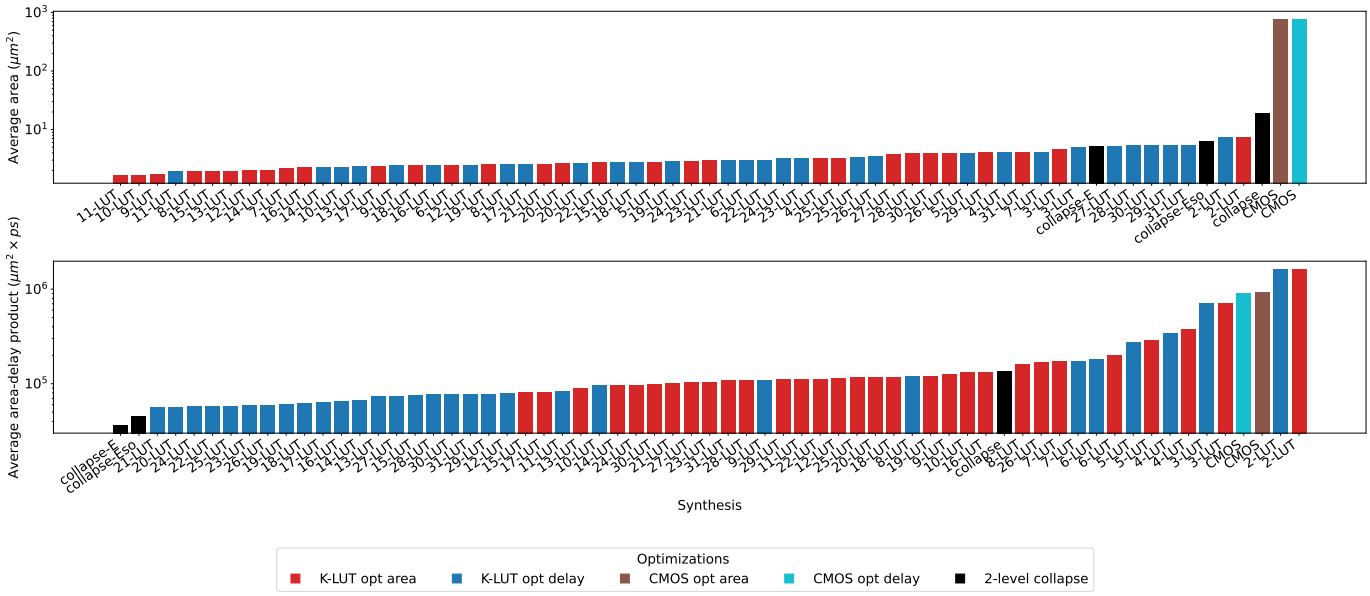

Fig. 9: Comparison w.r.t. CMOS in terms of average area and average area-delay product

[5] J. J. Y. et al., "Memristive devices for computing," *Nature nanotechnology*, vol. 8, 2013.

[6] J. R. B. et al., "Variation-tolerant computing with memristive reservoirs," *IEEE/ACM International Symposium in Nanoscale Architectures (NANOARCH) pp*, vol. 1, 2013.

[7] K. h. Kim et al., "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, 2011.

[8] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1718–1725.

[9] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, Oct 2015, pp. 335–342.

[10] E. Testa *et al.*, "Logic synthesis for established and emerging computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 165–184, Jan. 2019. [Online]. Available: https://doi.org/10.1109/jproc.2018.2869760

[11] M. Traiola *et al.*, "XbarGen: a memristor based boolean logic synthesis tool," in *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Sept 2016, pp. 1–6.

[12] "Xbargen website," https://github.com/mtraiola/XbarGen, accessed: 2017-09-18.

[13] S. Duan *et al.*, "Memristor-based cellular nonlinear/neural network: Design, analysis, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 6, pp. 1202–1213, June 2015.

[14] W. Wen *et al.*, "An EDA framework for large scale hybrid neuromorphic computing systems," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[15] A. Shafiee *et al.*, "ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 14–26.

[16] S. Kvatinsky *et al.*, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, Oct 2014.

[17] X. Wang *et al.*, "Synthesis of memristive circuits based on stateful IMPLY gates using an evolutionary algorithm with a correction function," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2016, pp. 97–102.

[18] A. Raghuvanshi *et al.*, "Logic synthesis and a generalized notation for memristor-realized material implication gates," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 470–477.

[19] S. Kvatinsky *et al.*, "MAGIC - memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, Nov 2014.

[20] S. Chakraborti *et al.*, "BDD based synthesis of boolean functions using memristors," in *2014 9th International Design and Test Symposium (IDT)*, Dec 2014, pp. 136–141.

[21] S. Shirinzadeh *et al.*, "Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 948–953.

[22] H. A. D. Nguyen *et al.*, "Synthesizing HDL to memristor technology: A generic framework," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2016, pp. 43–48.

[23] L. Xie *et al.*, "A mapping methodology of boolean logic circuits on memristor crossbar," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[24] N. Weste *et al.*, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.

[25] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, 2005.

[26] A. Mishchenko *et al.*, *ABC: A system for sequential synthesis and verification*, 2012. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

[27] R. K. Brayton *et al.*, *Logic Minimization Algorithms for VLSI Synthesis*. Springer US, 1984. [Online]. Available: https://doi.org/10.1007/978-1-4613-2821-6

[28] M. Traiola *et al.*, "Estimating dynamic power consumption for memristor-based CiM architecture," *Microelectronics Reliability*, vol. 80, pp. 241–248, Jan. 2018. [Online]. Available: https://doi.org/10.1016/j.microrel.2017.12.009

[29] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," MCNC Technical Report, Tech. Rep., Jan. 1991.

[30] K. McElvain, "IWLS'93 Benchmark Set: Version 4.0," Tech. Rep., May 1993.

[31] F. Brglez *et al.*, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," in *IEEE International Symposium Circuits and Systems (ISCAS'85)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[32] ——, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS'89)*, May 1989, pp. 1929–1934 vol.3.

[33] F. Corno *et al.*, "RT-level ITC'99 benchmarks and first ATPG results," in *IEEE Design Test of Computers*, Jul 2000, pp. 44–53 vol.17.

[34] L. Amaru, "The EPFL combinational benchmark suite," Integrated Systems Laboratory, EPFL, Lausanne, Switzerland, Tech. Rep., Sep. 2016. [Online]. Available: https://github.com/lsils/benchmarks

[35] C. Albrecht, "IWLS 2005 benchmarks," Tech. Rep., June 2005.

[36] "Freepdk45," https://eda.ncsu.edu/freepdk/freepdk45/, accessed: January 16, 2022.

[37] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.