SAID: A Supergate-Aided Logic Synthesis Flow for Memristive Crossbars

Valerio Tenace[§], Roberto G. Rizzo[§], Debjyoti Bhattacharjee[‡], Anupam Chattopadhyay[‡], Andrea Calimera[§]
[§]Department of Control and Computer Engineering, Politecnico di Torino, Italy
[‡]School of Computer Science and Engineering, Nanyang Technological University, Singapore

***Abstract*—A Memristor is a two-terminal device that can serve as a non-volatile memory element with built-in logic capabilities. Arranged in a crossbar structure, memristive arrays allow to represent complex Boolean logic functions that adhere to the logic-in-memory paradigm, where data and logic gates are glued together on the same piece of hardware. Needless to say, novel and *ad-hoc* CAD solutions are required to achieve practical and feasible hardware implementations. Existing techniques aim at optimal mapping strategies that account for Boolean logic functions described by means of 2-input NOR and NOT gates, thus overlooking the optimization capabilities that a smart and dedicated technology-aware logic synthesis can provide. In this paper, we introduce a novel library-free supergate-aided (SAID) logic synthesis approach with a dedicated mapping strategy tailored on MAGIC crossbars. Supergates are obtained with a Look-Up Table (LUT)-based synthesis that splits a complex logic network into smaller Boolean functions. Those functions are then mapped on the crossbar array as to minimize latency. The proposed SAID flow allows to (i) maximize supergate-level parallelism, thus reducing the total number of computing cycles, and (ii) relax mapping constraints, allowing an easy and fast mapping of Boolean functions on memristive crossbars. Experimental results obtained on several benchmarks from ISCAS'85 and IWLS'93 suites demonstrate that our solution is capable to outperform other state-of-the-art techniques in terms of speedup (3.89× in the best case), at the expense of a very low area overhead.**

I. Introduction

In a classical *von Neumann* architecture, a logic processor and a memory bank are two separate entities figuratively connected by a bi-directional stream of data. Since processor's frequencies have significantly increased over the past years, idle times in modern devices, due to memory synchronization, represent a huge portion of the computational effort [1]. As such a difference in performance widens, the *memory wall* proper of the *von Neumann* architecture is becoming a critical bottleneck. The pervasiveness of high-computing applications even at the consumer level, e.g., deep learning on mobile devices with face recognition or data analytics with smart objects via edge computing [2], has pushed the researchers to find an applicable solution at the intersection between logic and memory domains: the logic-in-memory (LiM) paradigm, where logic functionalities and data are hosted together on the same piece of hardware. The trigger that allowed LiM to succeed is the discovery of a new electronic element.

Initially theorized in 1971 [3], and fabricated for the first time in 2008 [4], the memristor (portmanteau of memory and resistor) is the fourth fundamental circuit element after the resistor, the inductor, and the capacitor. The name itself suggests some of the unique properties of the device. A memristor relates magnetic flux and charge with a current-voltage characteristic that exhibits a pinched hysteresis loop that enables the switching of resistive states. Such states are controlled by applying an appropriate voltage supply across the memristor. An important characteristic of memristors concerns state retention; indeed, the resistance value does not change even if the voltage supply is withdrawn. Therefore, memristors are suitable candidates to represent non-volatile, energy efficient, and high-density memory elements that can serve to store information with enhanced logic capabilities [5], where logic values are represented with distinct resistive states of the memristor. When connected by means of a lattice structure, memristors create a two-dimensional grid of devices, referred to as *crossbar*. By means of appropriate voltage levels applied on both rows and columns of the crossbar array, memristors are selected and connected together in order to carry out logic operations. Needless to say, whenever a new computational paradigm is introduced there is a call for novel CAD methodologies.

A. Motivation & Contributions

Different crossbar implementations rely on different logic primitives. First contributions include [6], [7], where memristors are configured to realize the material implication (IMPLY) function. Crossbar mapping was achieved by means of Binary Decision Diagrams (BDDs) [6], and Or-Inverter Graphs [7]. More recently, the ReVAMP crossbar implementation that leverages 3-input majority (MAJ) gates and a single-input inverter was proposed in [8]. Tailored on such architecture, a delay-optimal technology mapping algorithm was proposed in [9], along with complete technology mapping flows for crossbar-based computation of arbitrary functions described through MAJ-Inverter Graphs (MIGs) [10], [8]. It is worth to notice that both IMPLY- and MIG-based solutions rely on custom data structures that are usually not part of a standard synthesis flow. On the opposite, the MAGIC solution [11] proposes a NOR-INV representation of Boolean logic functions derived from classical AND-INV Graphs (AIGs) where nodes are mapped on 2-input NORs with appropriately negated edges. The flow takes as an input a mapped netlist obtained through the ABC tool [12] and technology mapping is carried out by means of a constrained theorem-proof solver. No technology-specific logic synthesis optimization was proposed

978-3-9819263-2-3/DATE19/©2019 EDAA

in this work, and the complexity of the mapping algorithm represents a serious concern in terms of scalability. A partial improvement has been introduced in [13] with a mapping strategy that relies on a simulated annealing algorithm. Obtained circuits are area and delay efficient thanks to the use of copy operation for partial results alignment. However, also in this case, no technology-aware optimization was exploited.

An important observation about crossbar mapping is that traditional optimization metrics used for logic synthesis do not directly translate into improvements for LiM implementations. For example, the depth of a logic network is not representative of the overall delay of the corresponding LiM circuit mapped on MAGIC crossbars. This rises the need for technology aware synthesis and optimizations that can augment the technology mapping procedures for significant improvements in terms of mapping strategies. Recently, technology-aware synthesis techniques have been introduced to reduce delay and area required for LiM computations using MIGs for ReRAM crossbar arrays that realize the ReVAMP architecture [14]. However, no such work exists in the context of MAGIC-based circuits. This is where this paper is positioned. Our intuition is that a small portion of the crossbar array can be used to map circuit's cuts described by means of *look-up tables* (LUTs), therefore naturally adhering to the geometrical shape of the crossbar itself. We refer to each LUT as a *supergate*, since our approach is library-free. This means that each LUT has an arbitrary number of primary inputs and product terms, thus enabling the representation of even complex logic gates within the MAGIC crossbar. Our optimization objective is to minimize the total number of computational cycles required to achieve the final result, while minimizing the area overhead w.r.t. other solutions.

The contributions of this work can be summarized as follows: (i) since LUTs are usually tailored on a Sum-of-Products (SoP) representation, we formalize the description of LUTs by means of a *NOR-of-NORs* representation; (ii) we propose an efficient supergate-aided (SAID) logic synthesis methodology that accounts for different levels of optimizations; and (iii) we propose a dedicated mapping approach of combinational logic functions for MAGIC crossbars. Experimental results obtained on two different sets of open-source benchmarks demonstrate that the proposed technique is capable to achieve a remarkable 3.89× best average speedup with a low 31% area overhead.

II. Background

A. Memristor-Aided loGIC

Kvatinsky et al. [15] proposed a stateful logic, called Memristor-Aided loGIC (MAGIC), that realizes NOR functions using a single operating voltage V_G. Separate memristors are used to represent the inputs and the output of the NOR gate, being the output memristor set to a High Resistance State (HRS) initially. Since NOR is a functional-complete operator, any arbitrary Boolean function can be realized as a sequence of NOR gates. The operating voltage V_G is applied to the columns representing inputs, whereas the output column is grounded. Figure 1-a shows the realization of a 2-input NOR using MAGIC. Due to the properties of the crossbar array, multiple-input, e.g., three or more, NOR gates can also be implemented [15].

Memristors are arranged in a crossbar configuration, with multiple devices sharing common wordlines and bitlines. Figure 1-b shows a crossbar with 3 rows (wordlines) and 5 columns (bitlines). The primary advantage of MAGIC operations is the capability to perform multiple operations in parallel in a crossbar structure. To enable this feature, a few constraints have to be followed. First, each output must be mapped to a unique memristor, but inputs shared by two different gates can be mapped to the same memristor. Second, for each gate, input and output memristors must be in the same row or in the same column. The parallel execution of multiple NOR gates is then achieved whenever inputs and outputs of two NOR gates are aligned in the same rows or columns. As an example, Figure 1-b shows the configuration of the crossbar for two parallel 3-input NOR operations: $g_1 = a \mathbin{\overline{\vee}} b \mathbin{\overline{\vee}} c$ and $g_2 = d \mathbin{\overline{\vee}} e \mathbin{\overline{\vee}} f$, being $\overline{\vee}$ the NOR operator. Rows and columns not involved in logic operations are isolated by means of an isolation voltage V_{ISO}.

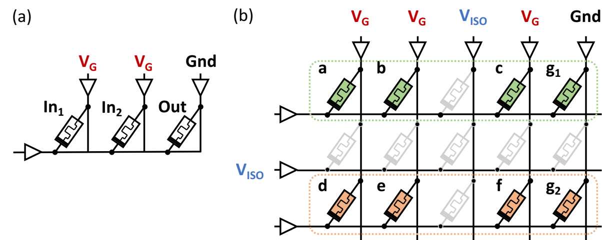

Figure 1: MAGIC crossbar. A two-input NOR gate (a), and two 3-input NOR gates configured for parallel execution (b).

B. NOR-of-NORs Supergate

In the context of LUT-based logic synthesis, a generic Boolean network $\mathcal{N}$ is represented by a directed acyclic graph $\mathcal{N} = (V, E)$. Each vertex $v \in V$ represents an arbitrary logic gate $\mathcal{L}$, i.e., a supergate in this work, and each directed edge $(u, v) \in E$ represents a connection from the output of node u to the input of node v.

Let us consider a LUT describing a simple Boolean function $\mathcal{L} : \mathbb{B}^3 \to \mathbb{B}$ defined as in (1), being $\wedge$ and $\vee$ the logical AND and OR operators, respectively.

$$\mathcal{L} = (a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \tag{1}$$

Function (1) is expressed in terms of SoP, therefore it cannot be directly mapped into a memristor crossbar that only performs NOR operations. However, it is possible to translate (1) in terms of the NOR operator [16] with the transformations reported in (2).

$$\begin{aligned} \mathcal{L} &= (\neg a \mathbin{\overline{\vee}} \neg b \mathbin{\overline{\vee}} c) \;\vee\; (a \mathbin{\overline{\vee}} \neg c) \\ &= \neg((\neg a \mathbin{\overline{\vee}} \neg b \mathbin{\overline{\vee}} c) \mathbin{\overline{\vee}} (a \mathbin{\overline{\vee}} \neg c)) \end{aligned} \tag{2}$$

More formally, a generic Boolean function $\mathcal{F} : \mathbb{B}^n \rightarrow \mathbb{B}$ expressed in SoP format can be expressed in terms of *NOR-of-NORs* (NoN) with the following three transformations:

1) Replace $\wedge$ and $\vee$ operations with $\overline{\vee}$;
2) Flip the polarity of each primary input;
3) Negate the result.

In practice, if we represent the original SoP-based LUT of function $\mathcal{L}$ as in Figure 2-a, then the equivalent NoN-based LUT representation is the one reported in Figure 2-b. Such

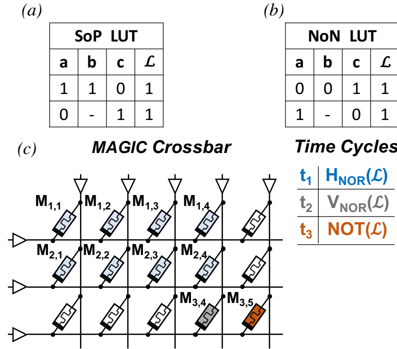

Figure 2: NOR-of-NORs LUT mapped onto MAGIC crossbar.

a LUT representation can then be mapped one-to-one on the memristive crossbar, as Figure 2-c suggests. Starting from the topmost left memristor ($M_{1,1}$), the LUT is mapped row-wise: $\{M_{1,1}, M_{1,2}\ M_{1,3}\}$, and $M_{1,4}$ (blue shaded regions) represent the inputs and output of the first term of the LUT. $\{M_{2,1}$, $M_{2,2}$, $M_{2,3}\}$, and $M_{2,4}$ are the inputs and output of the second term. Since inputs and outputs of the two NoN-based LUT terms adhere to the rule *same column, different row* [15], both can be executed in parallel in one cycle, t_1; we defined this basic LUT operation as horizontal NOR ($H_{NOR}(\mathcal{L})$) given the row-wise representation of the terms. The results of the two LUT terms, stored respectively in $M_{1,4}$ and $M_{2,4}$, are fed as inputs to the NOR gate composed of $\{M_{1,4}, M_{2,4}, M_{3,4}\}$; this operation replaces the original OR between product terms in classical SoP representations, and it is named Vertical NOR ($V_{NOR}(\mathcal{L})$). $V_{NOR}(\mathcal{L})$ takes another computational cycle (t_2) and its result is stored in $M_{3,4}$ (grey color). Lastly, the output of function $\mathcal{L}$ is computed by negating the result in $M_{3,4}$ to $M_{3,5}$ (orange color) during the last cycle (t_3). This operation is referred to as NOT($\mathcal{L}$). Therefore, the function $\mathcal{L}$ requires ten memristors and three computational cycles. To be noted that a function $\mathcal{L}$ with more than two LUT terms employs again three computation cycles, because all H_{NOR} operations can be executed in parallel. In addition, it is worth to notice that don't care conditions cannot be ignored due to the geometrical constraints that allow to carry out parallel execution of NOR operations. However, the power of don't care condition allows to reduce the number of realignment copies required in cascading LUTs. A detailed discussion on this particular aspect is reported in the following.

III. Supergate-Aided Logic Synthesis Flow

Efficient design automation for LiM imposes a dedicated technology-aware synthesis and optimization design flow, plus a fine-tuned mapping scheme that is able to exploit the expressive power of crossbar architectures. The proposed SAID flow encompasses three stages, as depicted in Figure 3, described as follows.

1. Logic Synthesis and Optimization: a logic circuit is synthesized, optimized, and mapped with supergates through the ABC k-LUT mapper that leverages the notion of priority cuts. Different implementations of multiple-input NOR gates are investigated as to find the best supergate configuration.

2. Crossbar Mapping: found solutions are mapped on the memristive crossbar through a dedicated mapping approach which exploits the geometrical shape of supergates; the result of this stage is a design space of logic circuit mappings.

3. Optimal Mapping Extraction: the mapping design space is explored through a Pareto analysis in order to extract the solution that minimizes both latency and crossbar area occupation, i.e., it provides the optimal mapping.

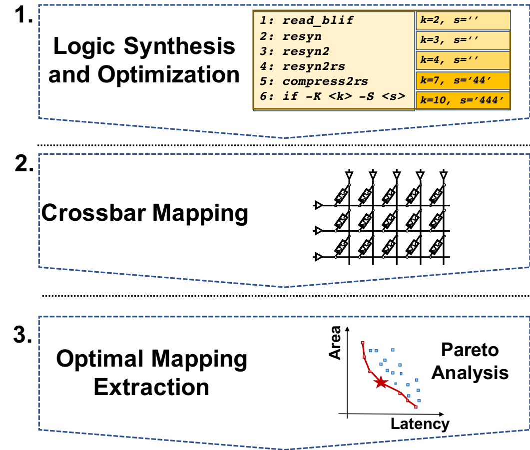

Figure 3: SAID Design Flow.

A. Logic Synthesis and Optimization

The proposed SAID approach runs optimization strategies for k-LUT synthesis. Figure 3 reports the adopted synthesis flow. In particular, the commands *resyn*, *resyn2*, *resyn2rs*, and *compress2rs* are heuristics methods, provided by the ABC synthesis tool, that reduce the cardinality of the AIG that describes the logic network. The resulting netlist is then mapped onto k-input supergate (k-S) cells with the *if* command (ABC k-LUT mapper), being $k = \{2, 3, 4, 7, 10\}$. This flow generates as many netlists as the number of k values, filling a design space which will be fed to the crossbar mapping. As remarked in Section II, MAGIC enables n-input NORs as long as defined voltage supply constraints are matched. Although the verification of this electrical condition is out of the scope of this work, we impose that the maximum feasible number of inputs cannot exceed 4. For this purpose, supergates with k equal to 7 and 10 have two dedicated architectures as shown

in Figure 4. This parameter choice fixes the maximum number of LUT terms for each supergate to 16, which means that the final NOR that evaluates all terms should theoretically be a 16-input NOR gate. However, in our experiments the maximum number of terms per supergate did not exceed 8.

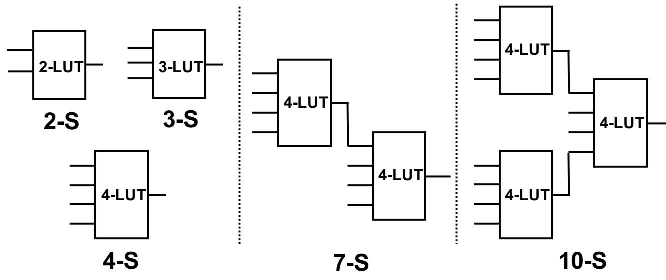

Figure 4: k-input supergate cells.

B. Crossbar Mapping

The entire set of solutions in the synthesis design space is fed to the technology mapping scheme. As a result of this stage, a design space of supergate-based logic circuits mapped on the MAGIC crossbar is generated.

The crossbar mapping scheme is implemented taking into account how logic supergates are distributed in the logic levels. The mapping of the supergates is performed row-wise, level by level as reported in Figure 5. The crossbar is divided in tiles which host the supergates. A tile is identified by (l, y) coordinates that represent respectively the l-th logic level of the netlist, and the vertical index of the tile (starting from the top). One supergate is allocated to one tile; as an example, $S_{1,1}$ is the supergate allocated to the tile *(1, 1)* of the crossbar. The mapping algorithm starts stacking the supergates of the first level sorted by their size, i.e., the number of their LUT terms, in a decreasing order. The mapping of cascading supergates is constrained to a dimensional condition which imposes that two supergates of different levels having greater or equal size cannot be placed at the same vertical position y. Although it could seem a limitation, the SAID mapping scheme relies on the following rule of thumb: *as the logic level increases, the supergate size decreases*.

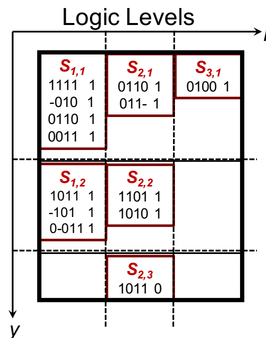

Figure 5: Crossbar tiles partitioning.

The SAID crossbar mapping is performed according to specific spatial rules, as depicted in Figure 6. Each supergate has a dedicated output-to-input alignment row (golden rows in Figure 6). It is used to relocate the output position w.r.t. cascading supergates. The output-to-input alignment is performed through a sequence of two NOT operations (double arrows in Figure 6) using an intermediate temporary memristor; alternatively, if a negated input is required by any of the LUT terms, the alignment is done with a single NOT operation (dashed arrow). Supergates in all levels must be separated at least by one row of memristors in order to avoid the overlapping of alignment rows, as shown for $S_{i+1,1}$ and $S_{i,1}$.

A 1-bit full adder mapped with the proposed scheme is reported in Figure 7. Firstly, $S_{1,1}$ and $S_{1,2}$ are placed separated by a crossbar row. Then, the algorithm tries to place $S_{2,3}$, the largest supergate in level 2; once verified that the size of $S_{2,3}$ is larger or equal to $S_{1,1}$ and $S_{1,2}$, $S_{2,3}$ is mapped below $S_{1,2}$, namely, the smallest supergate of the previous level. Lastly, as $S_{3,3}$ has the same size of $S_{2,1}$ and $S_{1,2}$, the first available vertical position is the one next to $S_{2,3}$.

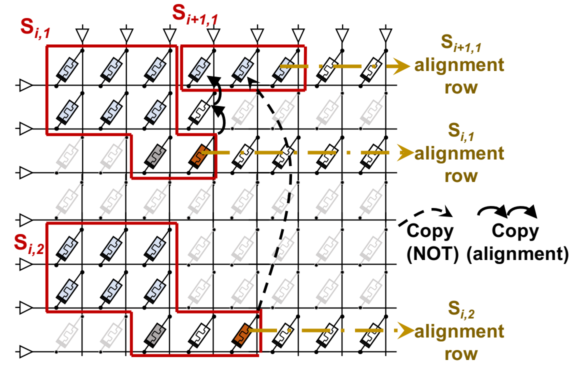

Figure 6: Spatial rules for SAID mapping.

Figure 7 also shows an example of how the SAID-based full adder works, reporting the logic operation per cycle. The sequence of the crossbar operations is previously loaded in memory and executed by the crossbar scheduler. Firstly, the scheduler sets the primary inputs (PI) of each supergate. The time needed for the scheduler to place each PI is not included in the total count of the computation cycles.

The operations of the first logic level are executed in the first 4 cycles: the LUT terms of all supergates are computed with the $\text{H}_{NOR}(S_{1,1},\ S_{1,2})$ operation; then the $\text{V}_{NOR}(S_{1,1},\ S_{1,2})$ operation in cycle 2 computes the NOR of the results of LUT terms in $S_{1,1}$; the NOT operation in cycle 3 extracts the output value for $S_{1,1}$, whereas in cycle 4 a NOT operation runs on $S_{1,2}$, which will be used in Level 3 to compute C_{out}. To be noticed that, having a single row, $S_{1,2}$ has already generated its output value at cycle 1.

After the execution of level 1, an output-to-input alignment is needed. The negated output of $S_{1,1}$ is aligned to the inputs of $S_{2,1}$ and to the first LUT term of $S_{2,3}$. The second LUT term of $S_{2,3}$ needs $S_{1,1}$, thus a two-cycle alignment is needed (from cycles 7 to 8).

At this point, supergates in level 2 are executed (9 to 12 cycles); the output S is obtained in this level from $S_{2,3}$. After

the alignment operations (13 to 16 cycles), $S_{3,3}$ is executed in only one cycle (the 17th), generating C_{out}.

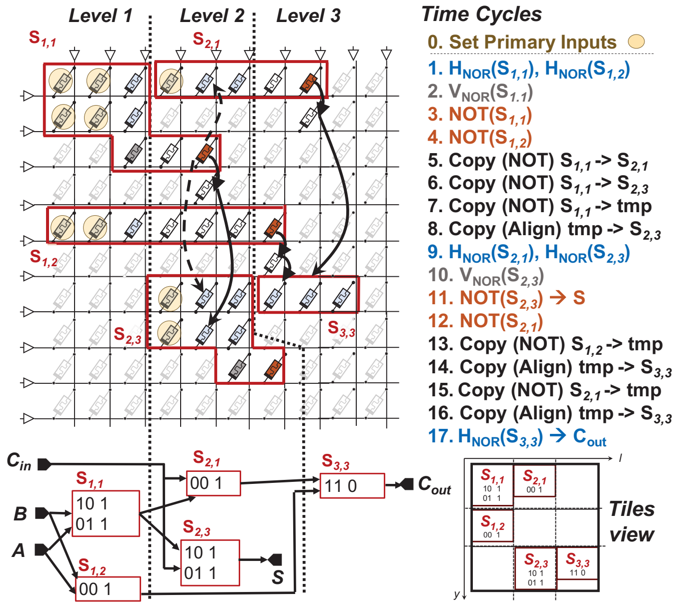

Figure 7: SAID-based implementation of a 1-bit full adder.

When a don't care condition appears in a supergate row, no realignment copies are required in cascading supergates, since the value is fixed to a logic "0" or "1" by the scheduler. Thus, a larger k value leads to more don't care conditions, improving the quality of the computation.

C. Optimal Mapping Extraction

The design space is inspected through a Pareto analysis which solves a multi-objective optimization problem. The best solution is the one that minimizes the area occupation, i.e., the number of memristors required for in-memory computing, and the latency represented by the number of time steps needed to compute all supergates. In particular, by varying the parameter k, the best solution in terms of the area/latency trade-off is determined.

IV. Experimental results

The proposed SAID logic synthesis reduces the latency by concurrently executing multiple LUTs of a logic level. However, this comes at the price of a higher area due to the redundancy introduced by supergates. In this section we quantify the efficiency of our approach w.r.t. state-of-the-art solutions. Since most solutions are not publicly available, we resort to experimental results provided by authors on their respective papers. Fortunately enough, most works share the same circuit benchmarks: the ISCAS'85 suite. Moreover, the work in [13] also provides results for an additional suite, the IWLS'93. Therefore, we separate our discussion depending on the benchmark suite.

Selected previous works are described as follows.

Talati et al. [17]: the first mapping algorithm for MAGIC crossbars. Mapped circuits are first synthesized with the `ABC` logic synthesis tool, and mapped on the crossbar with a heuristic approach.

Gharpinde et al. [18]: a NOR/INV netlist is mapped on MAGIC by replicating specific logic levels or gates in order to achieve the maximum parallelism still guaranteeing a square-shape allocation of memristors.

Thangkhiew et al. [13]: a simulated annealing algorithm is in charge of finding an optimal mapping of a NOR/INV netlist guaranteeing the minimum number of execution cycles.

A work that uses theorem-proving solvers for optimal mapping was recently proposed [11]. Due to the large computational effort required by the theorem prover (it exceeded a 48-hour timeout on most benchmarks), we could not evaluate the solution on the current benchmark set. Therefore, it could not be included in the comparison.

The adopted logic synthesis methodology and the proposed mapping algorithm for SAID-based crossbars are illustrated in Section III. The entire flow is executed by means of an in-house Python package that leverages the `ABC` tool.

Table I and Table II report the obtained experimental results for the ISCAS'85 and the IWLS'93 benchmark suites, respectively. In particular, for each implementation we compare the total number of cycles, or time steps, required to evaluate all the gates of the circuit (**Cycles** column), and the number of active memristors required to map the synthesized circuits (**Mems** column). Since all memristors are of equal size, comparing the number of devices for each implementation guarantees a fair comparison. Please note that the number of cycles for SAID-based solutions also include realignment cycles, i.e., the portion of time steps required to realign supergate inputs. Reported speedup and overhead values are all w.r.t. the SAID solution. Moreover, for each SAID-based benchmark we also report the best k-LUT configuration (please refer to Section III) found with the Pareto analysis (column **Best** k-S). Concerning the ISCAS'85 results, the highest average speedup of 3.31× is achieved w.r.t. [17]. In addition, an average area saving of 28% is also achieved. A favorable trade-off between speed and area is obtained w.r.t. [18] with an average 1.84× speedup and a negligible area overhead of only 7%. Lastly, the comparison with [13] demonstrates that our approach to the problem is more efficient than specific meta-heuristic mapping solutions (speedup of 1.7× with 11% more area), motivating, once more, how the secret of an optimal mapping on crossbar arrays lies in the logic synthesis stage, rather than in post-synthesis optimizations of logic networks obtained via trivial mappings. A particular mention is required for the `c6288` benchmark. Its SAID-based implementation does not allow to achieve any improvement. The motivation is in the structure of the benchmark itself: being an array of multiplier, it is composed of several full adder modules that could not be minimized with the adopted optimization flow. As a final remark, the IWLS'93 experiments demonstrate the scalability of the proposed approach. On average, SAID-based crossbar circuits result to be 3.89× faster, paying a low 31% area overhead. A last observation is related to the best k-LUT configuration. Indeed, in most cases, the best solution is found

	SAID			Talati et al. [17]				Gharpinde et al. [18]				Thangkhiew et al. [13]			
	Best k-S	**Cycles**	**Mems**	**Cycles**	**Mems**	**Speedup**	**Overhead**	**Cycles**	**Mems**	**Speedup**	**Overhead**	**Cycles**	**Mems**	**Speedup**	**Overhead**
c432	7	156	631	663	699	4.25	0.90	349	443	2.24	1.42	265	372	1.70	1.70
c499	4	420	1399	964	1005	2.30	1.39	1155	1293	2.75	1.08	935	1085	2.23	1.29
c880	3	482	1113	1063	1123	2.21	0.99	761	919	1.58	1.21	750	886	1.56	1.26
c1355	3	554	1182	2036	2077	3.68	0.57	1072	1232	1.94	0.96	938	1076	1.69	1.10
c1908	3	627	1095	2668	2701	4.26	0.41	1056	1150	1.68	0.95	970	1061	1.55	1.03
c2670	4	643	1249	3343	3576	5.20	0.35	1490	2018	2.32	0.62	1401	1915	2.18	0.65
c3540	4	1566	3261	4166	4216	2.66	0.77	2396	2554	1.53	1.28	2418	2663	1.54	1.22
c5315	4	1754	2937	7091	7269	4.04	0.40	3295	3737	1.88	0.79	3239	3795	1.85	0.77
c6288	2	4069	6067	2928	2960	0.72	2.05	3776	3952	0.93	1.54	5007	5344	1.23	1.14
c7552	3	2565	4003	9787	9994	3.82	0.40	3926	4514	1.53	0.89	3824	4415	1.49	0.91
Average		**1283.60**	**2293.70**	**3470.90**	**3562.00**	**3.31**	**0.82**	**1927.60**	**2181.20**	**1.84**	**1.07**	**1974.70**	**2261.20**	**1.70**	**1.11**

Table I: Synthesis and Mapping Results on ISCAS'85 benchmarks

	SAID			Thangkhiew et al. [13]			
	Best k-S	**Cycles**	**Mems**	**Cycles**	**Mems**	**Speedup**	**Overhead**
5xp1	4	106	251	228	244	2.15	1.03
9sym	7	160	1026	552	594	3.45	1.73
alu4	7	1124	5331	2681	2794	2.39	1.91
apex1	7	1764	3170	5251	5547	2.98	0.57
apex2	7	241	880	667	736	2.77	1.20
apex4	4	4477	5050	7105	7442	1.59	0.68
apex5	7	777	2223	1966	2319	2.53	0.96
b12	10	26	283	137	169	5.27	1.67
bw	10	53	718	367	385	6.92	1.86
clip	7	135	451	239	261	1.77	1.73
con1	10	5	52	35	48	7.00	1.08
cordic	7	28	164	117	164	4.18	1.00
duke2	10	300	1632	1261	1342	4.20	1.22
e64	10	134	394	1394	1647	10.40	0.24
ex1010	7	2104	9945	6947	7290	3.30	1.36
ex5p	10	277	2061	1229	1286	4.44	1.60
inc	10	55	280	264	282	4.80	0.99
misex1	10	17	184	141	157	8.29	1.17
misex2	10	30	125	239	287	7.97	0.44
misex3c	7	518	2551	2976	3094	5.75	0.82
misex3	10	1053	6184	1494	1563	1.42	3.96
pdc	7	1959	8403	2142	2237	1.09	3.76
rd53	4	31	83	75	86	2.42	0.97
rd73	4	150	379	262	280	1.75	1.35
rd84	4	252	745	410	428	1.63	1.74
sao2	10	79	559	309	331	3.91	1.69
seq	7	1957	7792	4658	4884	2.38	1.60
spla	7	1050	4036	2312	2399	2.20	1.68
squar5	10	15	98	101	112	6.73	0.88
t481	4	30	106	114	140	3.80	0.76
table3	3	1766	2242	4256	4418	2.41	0.51
table5	4	1169	2009	3851	4028	3.29	0.50
vg2	10	55	280	289	345	5.25	0.81
xor5	3	12	35	22	31	1.83	1.13
Average		**644.38**	**2050.65**	**1590.91**	**1687.35**	**3.89**	**1.31**

Table II: Synthesis and Mapping Results on IWLS'93 benchmarks

with $k \geq 3$. This demonstrates that the intuition of using multiple-input NOR gates is a viable solution to improve the figures of merit of LiM applications.

V. Conclusions

In this work we presented a novel logic synthesis approach with a dedicated mapping strategy for logic functions mapped on MAGIC crossbars. With a best average speedup of $3.89\times$, the proposed solution is capable to guarantee a fast execution of rather complex logic functions at the expense of a low area overhead (31% on average).

References

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[2] S. Yao *et al.*, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 351–360.

[3] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[4] D. B. Strukov *et al.*, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[5] E. Linn *et al.*, "Beyond von neumannlogic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

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

[7] A. Chattopadhyay and Z. Rakosi, "Combinational logic synthesis for material implication," in *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*. IEEE, 2011, pp. 200–203.

[8] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, "ReVAMP: ReRAM based VLIW architecture for in-memory computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 782–787.

[9] D. Bhattacharjee and A. Chattopadhyay, "Delay-optimal technology mapping for in-memory computing using reram devices," in *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 2016, pp. 1–6.

[10] M. Soeken *et al.*, "An MIG-based compiler for programmable logic-in-memory architectures," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[11] R. B. Hur *et al.*, "Simple magic: synthesis and in-memory mapping of logic execution for memristor-aided logic," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 225–232.

[12] B. L. Synthesis and V. Group, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/~alanmi/abc/, 2016.

[13] P. L. Thangkhiew and K. Datta, "Scalable in-memory mapping of boolean functions in memristive crossbar array using simulated annealing," *Journal of Systems Architecture*, vol. 89, pp. 49–59, 2018.

[14] D. Bhattacharjee, L. Amaŕu, and A. Chattopadhyay, "Technology-aware logic synthesis for reram based in-memory computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 1435–1440.

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

[16] V. Tenace *et al.*, "Multi-function logic synthesis of silicon and beyond-silicon ultra-low power pass-gates circuits," in *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[17] N. Talati *et al.*, "Logic design within memristive memories using memristor-aided logic (magic)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[18] R. Gharpinde *et al.*, "A scalable in-memory logic synthesis approach using memristor crossbar," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 2, pp. 355–366, 2018.