COMPACT: Flow-Based Computing on Nanoscale Crossbars with Minimal Semiperimeter

Sven Thijssen
Department of Computer Science
University of Central Florida
Orlando, USA
sven.thijssen@knights.ucf.edu

Sumit Kumar Jha
Department of Computer Science
University of Texas at San Antonio
San Antonio, USA
sumit.jha@utsa.edu

Rickard Ewetz
Department of ECE
University of Central Florida
Orlando, USA
rickard.ewetz@ucf.edu

***Abstract*—In-memory computing is a promising solution strategy for data-intensive applications to circumvent the *von Neumann bottleneck*. Flow-based computing is the concept of performing in-memory computing using sneak paths in nanoscale crossbar arrays. The limitation of previous work is that the resulting crossbar representations have large dimensions. In this paper, we present a framework called COMPACT for mapping Boolean functions to crossbar representations with minimal semiperimeter (the number of wordlines plus bitlines). The COMPACT framework is based on an analogy between binary decision diagrams (BDDs) and nanoscale memristor crossbar arrays. More specifically, nodes and edges in a BDD correspond to wordlines/bitlines and memristors in a crossbar array, respectively. The relation enables a function represented by a BDD with n nodes and an odd cycle transversal of size k to be mapped to a crossbar with a semiperimeter of n+k. The k extra wordlines/bitlines are introduced due to crossbar connection constraints, i.e. wordlines (bitlines) cannot directly be connected to wordlines (bitlines). For multi-input multi-output functions, COMPACT can also be applied to shared binary decision diagrams (SBDDs), which further reduces the size of the crossbar representations. Compared with the state-of-the-art mapping technique, the semiperimeter is reduced from $2.13n$ to $1.09n$ on the average, which translates into crossbar representations with 78% smaller area. The power consumption and the computation delay are on the average reduced by 7% and 52%, respectively.**

***Index Terms*—flow-based, in-memory, computing, memristor, crossbar, synthesis**

I. Introduction

Many modern computer architectures are based on the concepts defined in *First draft of a report on the EDVAC* by von Neumann [1]. These computer architectures suffer from the *von Neumann bottleneck*. This bottleneck is an inevitable consequence of the data transfer between separated memory units and processing units. [2]. The in-memory computing paradigm aims to solve this bottleneck by unifying memory storage and computation.

In 1971, L. Chua introduced a new circuit element, which he called *memristor* [3]. In 2008, Hewlett Packard Laboratories was the first to finally develop a physical model of this fourth fundamental circuit element [4]. This led to the development of new computing paradigms using memristors, such as *material-based implication logic* (IMPLY) [5], *memory-aided logic (MAGIC)* [6] and *flow-based computing* [7]. Each of these approaches have their respective strengths and weaknesses. For IMPLY-based logic, a major drawback is the number of complex computational steps required to synthesize a Boolean function [8], [9]. On the other hand, the parallelism within the MAGIC-style is fundamentally limited.

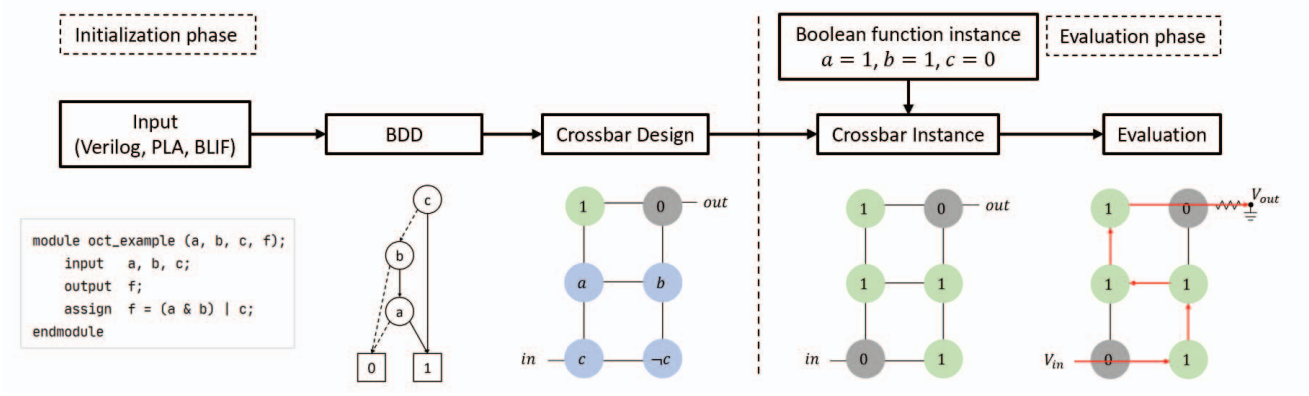
The flow-based computing paradigm is based on taking advantage of the natural flow of electrical current. By programming the resistance of memristors in a crossbar based on Boolean variables, Boolean functions can be evaluated by applying a high potential to the bottom most wordline and measuring the output current from a predefined wordline. The function evaluates to true if and only if there exists at least one path from the input to the output containing only memristors in the low resistive state.

Flow-based computing has been explored based on negation normal form (NNF) [7], disjunctive normal form (DNF), conjunctive normal form (CNF), simulated annealing [10] and satisfiability modulo theories (SMT) [11]. Unfortunately, these initial methods were computationally expensive or resulted in crossbar representations with large dimensions. To overcome these shortcomings, recent studies are based on mapping binary decision diagrams (BDDs) to crossbars using inductive staircase structures. The mapping of BDDs in the form of reduced ordered binary decision diagrams (ROBDD) and free binary decision diagrams (FBDD) has been explored [12], [13].

The staircase structures span from the bottom-left corner to the top-right corner of the crossbar. These inductive techniques are promising because both the number of rows and columns can be proved to grow linearly with the number of nodes in the BDD [12]. In particular, the dimensions of the nanoscale crossbar is upper bounded by $3n$ by n [12], where n is the number of nodes in the BDD. Fortunately, it can be observed that the crossbar representations have a dimension of closer to n by n in practice. Nevertheless, some rather simple Boolean functions still result in crossbars with excessive dimensions.

In this paper, we propose a framework called COMPACT for mapping BDDs into crossbar representations with minimal semiperimeter. The framework is based on an analogy between BDDs and nanoscale memristor crossbars. More specifically, nodes and edges in a BDD correspond to wordlines/bitlines and memristors in a crossbar, respectively. The relation enables a

This work was in part supported by NSF awards CCF-1755825, CNS-1908471, and CCF-1822976.



(a) Verilog code (b) ROBDD (c) Crossbar design $\mathcal{D}$ (d) Crossbar instance $\mathcal{I}$ (e) Evaluation $f = 1$

Fig. 1. Overview of the flow-based computing paradigm

BDD with n nodes and an odd cycle transversal of size k to be mapped into a crossbar representation with a semiperimeter of n+k. COMPACT determines such crossbar representations by viewing the BDD as a graph and formulating a node labeling problem. The node labeling problem is equivalent to an odd cycle transversal problem, which can be solved using a vertex cover formulation. We also observe that for multi-input multi-output functions, COMPACT can be directly applied to shared binary decision diagrams (SBDDs), which further improves the crossbar size. Compared with previous works, the experimental results show that the semiperimeter and area is reduced by 54% and 78% on average, respectively.

The remainder of the paper is organized, as follows: background in Section II, the BDD-crossbar analogy is given in Section III. The COMPACT framework is presented in Section IV and the extension to SBDDs in Section V. The paper is concluded with experimental results in Section VI.

II. Background

A. Binary decision diagrams

A BDD is a graph representation of a Boolean function. The internal nodes are Boolean variables and the leaf nodes are either '0' or '1'. A BDD is evaluated by traversing the nodes along a path from the root node to a leaf node. At each internal node of the BDD, depending on the value of the Boolean variable, one must decide which path to follow to evaluate an instance. [14] ROBDDs and FBDDs are extensions of BDDs for multi-input single-output functions that are optimized to minimize number of nodes and edges. BDDs can be extended to multi-input multi-output functions using shared binary decision diagrams (SBDDs). In SBDDs, multiple ROBDDs are merged together [15].

B. Nanoscale memristor crossbars

A memristor crossbar is a two-dimensional array consisting of two layers of nanowires. The horizontal nanowires are wordlines and the vertical nanowires are bitlines. Each layer is a set of parallel nanowires with each layer being perpendicular to one another. A memristor connects one layer with another at the intersections of the perpendicular nanowires [7]. Memristors with high endurance and fast switching speed have been demonstrated for memory applications [16]. A major concern for memory applications is the occurrence of currents on sneak paths, which reduces the effective write voltage [16]. In contrast, flow-based computing is based on leveraging the sneak paths to perform computation.

C. Flow-based in-memory computing

Flow-based computing is based on evaluating Boolean functions using the sneak currents that naturally occur in nanoscale crossbars. Computing within the paradigm is performed using a one-time costly initialization phase and an efficient and fast evaluation phase, which is illustrated in Figure 1.

In the initialization phase, a Boolean function f is converted into a crossbar representation $\mathcal{D}$. The Boolean function f is specified using a Verilog, BLIF or PLA file. A Boolean function $f = (a \wedge b) \vee c$ is shown in Figure 1(a). Next, a BDD representation of f is constructed using ABC/CUDD [17], which is shown in Figure 1(b). Previous work mainly utilized BDDs in the form of ROBDDs [12]. The next step is to map the BDD into a crossbar representation $\mathcal{D}$. This involves assigning each memristor in the crossbar to logical '0' or '1' or a Boolean variable $\{a,b,c\}$ or the negation of a Boolean variable $\{\neg a,\neg b,\neg c\}$. An input port and an output port are also assigned to the crossbar. A crossbar representation that realizes the BDD in Figure 1(b) is shown in Figure 1(c).

In the evaluation phase, the Boolean function is evaluated using the crossbar representation and an instance of the Boolean variables. The first step is to program the memristors in the crossbar based on the instance of the Boolean variables. Memristors in the crossbar are programmed to have *low* (*high*) resistance if the assigned logic expression is true (false). In the example, the crossbar instantiation for $a = 1$, $b = 1$ and $c = 0$ is shown in Figure 1(d). Next, an input voltage V_{in} is applied to the bottom most wordline and f is evaluated by measuring the output voltage V_{out} across a sensing resistor, which is connected to ground. In the example, it can be observed that there exists a path from the input to the output that only contains memristors with low resistance. Therefore, the output voltage is high and the Boolean function f evaluates to true, which is shown in Figure 1(e).

Problem formulation: The objective of this paper is to find the smallest *valid* crossbar representation $\mathcal{D}$ that realizes a Boolean function ϕ. In this paper, the size of the crossbar representations is evaluated in terms of semiperimeter (wordlines plus bitlines) and area (wordlines times bitlines). A crossbar representation $\mathcal{D}$ is a valid representation of a Boolean function ϕ if and only if for every instance of the Boolean variables, there exists a path from the input to the output using only memristors in the low resistive state when ϕ evaluates to true.

III. Analogy between BDDs and crossbars

The COMPACT framework in this paper is based on the observation that an analogy exists between BDDs and memristor crossbars. More specifically, the nodes and edges within a BDD correspond to the bitlines/wordlines and memristors in a crossbar, respectively. Theoretically, a BDD with n nodes can be mapped to a crossbar with a semiperimeter of n. However, a memristor crossbar places inherent constraints on the connections realized by the memristors; wordlines cannot be connected directly to wordlines and bitlines cannot be connected directly to bitlines. Therefore, extra hardware resources (intermediate bitlines or wordlines) are needed to realize such connections. One way to circumvent the connection constraint problem is to map each node to both a wordline and a bitline. However, this leads to a crossbar representation with a semiperimeter of $2n$. The COMPACT framework aims to find smaller crossbar designs by mapping as few nodes as possible to both wordlines and bitlines while resolving the connection constraints. In fact, COMPACT is capable of assigning the fewest possible BDD nodes to both wordlines and bitlines, which results in crossbar representations with minimal semiperimeter.

IV. The COMPACT framework

The flow of the COMPACT framework is shown in Figure 2 and illustrated with an example in Figure 3. The input to the framework is a Boolean multi-input single-output function represented using a ROBDD or a Boolean multi-input multi-output function represented using a SBDD, which is illustrated in Figure 3(a). The output of the framework is a crossbar representation $\mathcal{D}$ of the Boolean function.

The main steps of COMPACT are graph pre-processing, VH-labeling and crossbar mapping. In the graph pre-processing step, the BDD is converted into a graph representation. In the VH-labeling step, each node in the graph is assigned a label V, H or VH, indicating if they will be mapped to a vertical bitline (V), horizontal wordline (H), or both a vertical bitline and a horizontal wordline (VH). In the crossbar mapping step, nodes in the graph are bound to specific wordlines/bitlines according to the assigned labels. The edges in the graph are correspondingly assigned to memristors in the crossbar.

A. Graph pre-processing

In this section, the input BDD is converted into an undirected graph G. This is performed by first removing terminal node '0' and its incoming edges. The zero can be removed because flow-based computing aims to only capture the '1' output. Finally, the graph representation is obtained by mapping each node/edge in the BDD to an node/edge in an undirected graph. The resulting graph G of the BDD in Figure 3(a) is shown in Figure 3(b).

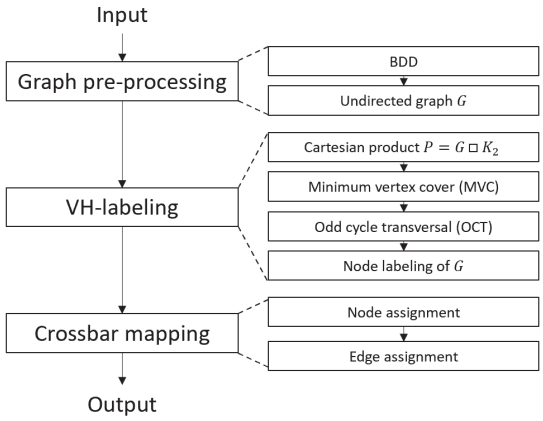


Fig. 2. Overview of the COMPACT framework

B. VH-labeling

The input to the VH-labeling step is the undirected graph G. The step involves assigning a label V, H, or VH to each node in the graph. The labels are introduced to ensure that all edges in the graph can later be realized using a memristor in the subsequent crossbar mapping step, i.e., preemptively handling the connection constraints. We first define the VH-labeling problem as a mathematical optimization problem in Section IV-B1. Next, we provide an optimal algorithm to solve the VH-labeling problem in Section IV-B2.

1) The VH-labeling problem: Let $G = (U, E)$ be a graph with a set of vertices U and a set of edges E. The VH-labeling problem consists of assigning a label $\{V,H,VH\}$ to each node in the graph such that the number of VH labels are minimized while satisfying the connection constraints. We formally define the VH-labeling problem as follows:

$$\begin{aligned} \min \quad & |\{v \,|\, v = L^{-1}(VH)\}| \\ \text{s.t.} \quad & \neg(L(u) = V \wedge L(v) = V), \qquad (u,v) \in E \\ & \neg(L(u) = H \wedge L(v) = H) \qquad (u,v) \in E \end{aligned} \tag{1}$$

where u and v are vertices in U. $L : U \to \{V, H, VH\}$ is the label given to node v.

The objective directly minimizes the number of VH labels, which explicitly defines the semiperimeter of the resulting crossbar representation. The semiperimeter is equal to n+k if the graph has n nodes and k VH labels. The area is implicitly optimized by minimizing the semiperimeter. The two constraints ensure that no adjacent nodes in the graph G are assigned (V,V) or (H,H) labels, as it would be impossible to connect the corresponding bitlines or wordlines using a memristor.

2) Solving the VH-labeling problem: In this section, we provide an optimal algorithm to solve the VH-labeling problem, which results in crossbar representations with the minimal semiperimeter.

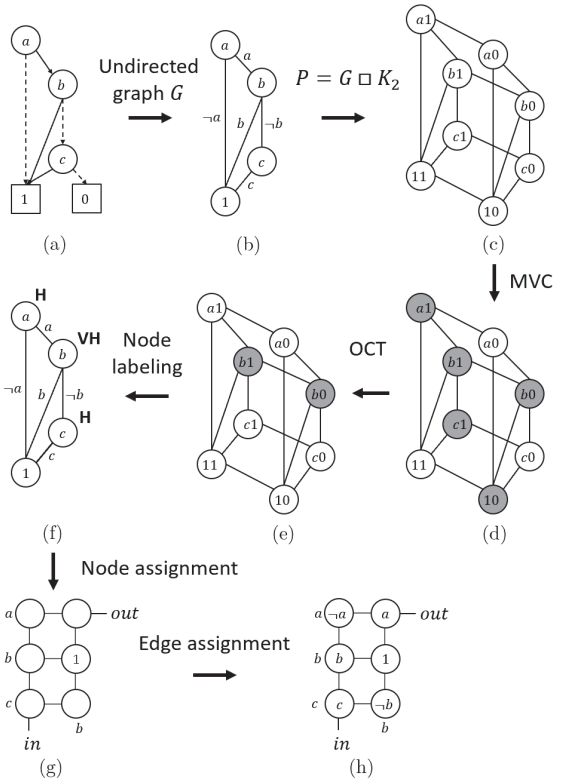


Fig. 3. Example of the COMPACT framework

If G is bipartite, it is trivial to determine an optimal solution to Eq (1) using 2-coloring. The colors would be the labels V and H. If G is not bipartite, no 2-coloring exists [18]. Hence, not every pair of adjacent nodes can be given a label V and H. Consequently, a VH label must be assigned to at least one node. A necessary condition for a graph to be bipartite is that it does not contain an odd-length cycle [18].

Our optimal solution to the VH-labeling problem lies in the observation that solving Eq (1) is equivalent to finding the largest induced bipartite subgraph G_B of the graph G. The nodes in G that are not part of G_B are the nodes labeled VH. The nodes in G_B can trivially be labeled V and H using 2-coloring. Moreover, finding the largest induced bipartite subgraph is equivalent to the odd-cycle transversal problem.

Definition 1 (Odd Cycle Transversal): The odd cycle transversal (OCT) of an undirected graph $G = (V, E)$ is a set $X \subseteq V$, $|X| \leq k$, such that $V - X$ is a bipartite graph. [19]

We use lemma 1 to find such odd cycle transversal for G.

Lemma 1: A graph $G = (V, E)$ with $|V| = n$ has an odd cycle transversal X, $|X| \leq k$, if and only if $P = G \square K_2$ has a vertex cover $VC(P)$ such that $|VC(P)| \leq n + k$. [19]

We leverage this solution method to finding a minimum vertex cover of P and thus to finding a smallest odd cycle transversal of G. In Figure 3(c), we show the graph $P = G \square K_2$, i.e., the Cartesian product of G with K_2. K_2 is a graph with two nodes connected by an edge. The resulting graph P contains two duplicates of graph G. Above this, a node's two duplicates are connected by an edge. If the nodes in K_2 are given a name 0 and 1, then the name of a node in P is the concatenation of the node's respective name in G and either 0 or 1. For example, node a in graph G is duplicated in two nodes, $a0$ and $a1$ in graph P. A vertex cover $W = VC(G)$ for a graph $G = (U, E)$ is a set of nodes $W \subseteq U$ such that for each edge $e = (u, v) \in E$ at least one node u or v is in W. The minimum vertex cover problem can be solved using integer linear programming (ILP) [20]. The minimum vertex cover of P in Figure 3(c) is shown in Figure 3(d). If both products v_0 and v_1 of a node v are present in the vertex cover W, then v belongs to the odd cycle transversal X of G. It can be observed that both $b0$ and $b1$ belong to the vertex cover in Figure 3(e), which results in that the node is part of the OCT and is labeled VH in Figure 3(f). Finally, the largest induced bipartite subgraph G_B is obtained by only considering the nodes in G which are not labeled VH. The labeling of G_B is performed using traditional 2-coloring, as shown in Figure 3(f).

C. Crossbar mapping

In the crossbar mapping step, we bind the graph G to a crossbar representation $\mathcal{D}$ according to the assigned labels, which ensures that the connection constraints can be satisfied.

The mapping is performed by using a node assignment step and an edge assignment step. In the node assignment step, each node in the graph is assigned to a bitline, wordline, or both a bitline and a wordline according to the label in the graph, i.e., nodes labeled V (H) are assigned to bitlines (wordlines). Nodes labeled VH are assigned to both a bitline and a wordline. However, these wordlines and bitlines are supposed to be connected. Therefore, we also program the memristor in the intersection of the corresponding wordline and bitline to have low resistance or '1'. The crossbar representation following the node assignment step is shown in Figure 3(g).

In the edge assignment step, each edge in the graph is mapped to a memristor in the crossbar such that it connects the bitlines and wordlines that correspond to the nodes in the graph. Following the node assignment step, the edge assignment step maps the variables and their negations onto the crossbar representation, as shown in Figure 3(h). The output is a crossbar design $\mathcal{D}$ for a Boolean function ϕ using the COMPACT framework.

V. Extension of COMPACT to SBDD

Previous work on flow-based computing for multi-input multi-output functions relied on splitting the function into many multi-input single-output functions. Next, each multi-input single-output function was converted into a BDD and mapped to a crossbar design. These separate crossbar designs can be placed in a single crossbar by aligning the designs along the diagonal. We observe that using COMPACT there is no need to specify a single function for each output. Instead, we can directly convert the multi-input multi-output function into a SBDD. Next, the SBDD can directly be mapped into a crossbar design. This may result in smaller crossbar designs as some parts of the single-output BDDs can be shared across multiple outputs.

TABLE I
COMPARISON OF FLOW-BASED COMPUTING ALGORITHMS IN TERMS OF NUMBER OF BDD NODES, ROWS, COLUMNS, SEMIPERIMETER, AREA AND SYNTHESIS TIME.

Benchmark	Chakraborty et al. with ROBDD [12]						COMPACT with ROBDD						COMPACT with SBDD					
	Nodes (num)	Rows (num)	Cols (num)	Semi (num)	Area (num)	Time (min)	Nodes (num)	Rows (num)	Cols (num)	Semi (num)	Area (num)	Time (min)	Nodes (num)	Rows (num)	Cols (num)	Semi (num)	Area (num)	Time (min)
parity	32	31	32	63	992	0.1	32	16	16	32	256	0.1	32	16	16	32	256	0.1
cm150a	33	32	48	80	1536	0.0	33	12	22	34	264	0.0	33	12	22	34	264	0.0
t481	33	32	37	69	1184	0.0	33	17	23	40	391	0.0	33	17	23	40	391	0.0
cm162a	53	48	67	115	3216	1.7	53	29	34	63	986	2.0	53	29	34	63	986	2.0
x2	59	52	74	126	3848	0.0	59	33	35	68	1155	0.1	59	33	35	68	1155	0.1
cm163a	79	66	90	156	5940	0.1	79	41	46	87	1886	0.1	51	25	31	56	775	0.0
misex1	79	72	92	164	6624	0.0	79	39	47	86	1833	0.1	48	21	29	50	609	0.0
cordic	101	99	131	230	12969	0.0	101	52	53	105	2756	0.0	81	42	44	86	1848	0.1
5xp1	127	117	155	272	18135	0.1	127	67	71	138	4757	0.1	89	52	53	105	2756	0.1
clip	153	148	186	334	27528	0.0	153	84	84	168	7056	0.0	153	84	84	168	7056	0.0
alu4	1289	1281	1461	2742	1871541	0.4	1289	683	686	1369	468538	1.5	1289	683	686	1369	468538	1.5
misex3	1566	1552	1670	3222	2591840	0.4	1566	845	858	1703	725010	4.7	1302	674	676	1350	455624	2.2
apex2	1647	1644	1645	3289	2704380	1.4	1647	909	936	1845	850824	1.1	1647	909	936	1845	850824	1.1
apex4	1662	1644	1789	3433	2941116	0.4	1662	806	945	1751	761670	0.4	1022	508	528	1036	268224	2.3
apex5	2759	2674	3591	6265	9602334	1.0	2759	1409	1497	2906	2109273	0.9	2759	1409	1497	2906	2109273	0.9
seq	3266	3231	3690	6921	11922390	1.5	3266	1743	1778	3521	3099054	1.1	3266	1743	1778	3521	3099054	1.1
Normalized	1.00	1.00	1.00	1.00	1.00	**1.00**	1.00	0.54	0.49	0.51	0.27	1.88	**0.89**	**0.48**	**0.44**	**0.46**	**0.22**	2.01

VI. EXPERIMENTAL EVALUATION

The COMPACT framework is implemented in Python and the experiments have been conducted on a machine with Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz, 2808 Mhz with 2 cores and 4 logical processors. We evaluate the effectiveness of the COMPACT framework using 16 combinational circuits from the RevLib benchmark suite [21]. A summary of the properties of the circuits is shown in Table II.

TABLE II
OVERVIEW OF INPUT CIRCUITS.

Benchmark	Inputs	Outputs	ROBDD Vertices	ROBDD Edges	SBDD Vertices	SBDD Edges
parity	16	1	32	60	32	60
cm150a	21	1	33	48	33	48
t481	16	1	33	58	33	58
cm162a	14	5	53	83	53	83
x2	10	7	59	89	59	89
cm163a	16	13	79	92	51	78
misex1	8	7	79	101	48	72
cordic	23	2	101	170	81	142
5xp1	7	10	127	185	89	162
clip	9	5	153	253	153	253
alu4	14	8	1289	2299	1289	2299
misex3	14	14	1566	2486	1302	2292
apex2	39	3	1647	2759	1647	2759
apex4	9	19	1662	2912	1022	1910
apex5	117	88	2759	4352	2759	4352
seq	41	35	3266	4982	3266	4982

Performance is evaluated in terms of hardware utilization, power consumption, synthesis time and computation delay. Hardware utilization is evaluated in terms of the crossbar dimensions, i.e. in terms of rows, columns, semiperimeter and area. Power consumption is proportional to the number of rows of the crossbar design. The synthesis time is the run-time of COMPACT in the one-time initialization phase. The computation delay is the number of time-steps required to evaluate the Boolean function in the evaluation phase. The number of time steps is equal to the number of rows plus one. One time step per wordline is required to program the devices [22] and one time step is required to evaluate the Boolean function. Note that we have verified that all the crossbar designs are *valid* using SPICE simulations and the memristor model in [22].

We compare COMPACT with the state-of-the-art flow-based computing algorithm in Section VI-A. We compare COMPACT with other in-memory computing paradigms in Section VI-B.

A. Comparison with previous work on flow-based computing

In this section, we compare COMPACT based on both ROBDDs and SBDDs with the state-of-the-art flow-based computing algorithm in [12]. We evaluate the performance in terms of hardware utilization in Table I. The computation delay and power consumption are evaluated in Figure 4.

The number of BDD nodes, the number of rows, columns, semiperimeter and area for each of the techniques is shown in Table I. It can be observed that the algorithm in [12] is capable of mapping all the input circuits into valid crossbar representations. The semiperimeter is approximately $2.13n$, where n is the number of BDD nodes. The run-time is less than two minutes for all circuits.

Compared with the algorithm in [12], COMPACT based on ROBDDs reduces the number of rows, columns, semiperimeter and area, with 46%, 51%, 49%, 73%, respectively. Smaller crossbar designs are obtained due to the fact that most BDD nodes are only mapped to a single wordline or bitline, whereas in the previous work all nodes are mapped to at least one wordline and one bitline. The semiperimeter is approximately $1.09n$, which demonstrates that only 9% of the nodes in the BDD are labeled VH and are mapped to both a wordline and bitline. COMPACT based on SBDDs reduces the the rows, columns, semiperimeter and area by 52%, 56%, 54% and 78%, respectively. This stems from that the number of nodes required to represent a function using a SBDD instead of multiple ROBDDs is smaller, as the semiperimeter is $1.09n$, where n again is the number of BDD nodes. Note that merging the ROBDDs for the different outputs does not always generate a smaller SBDD. This explains why the same results are obtained for both ROBDD and SBDD on several circuits.

Figure 4 shows the normalized power and computation delay for the three methods across all the circuits. It can be observed that COMPACT based on ROBDDs and the algorithm in [12] result in the same power consumption. This stems from that the number of memristors that are required to be programmed

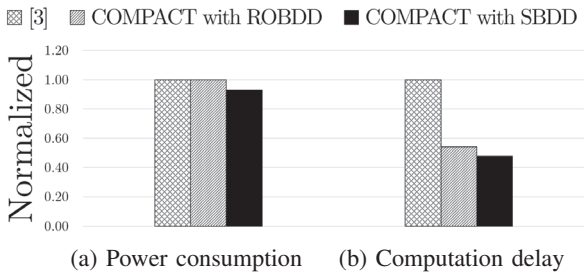


Fig. 4. Flow-based computing

is equal to the number of edges in the ROBDD, which is shown in Table II. COMPACT based on SBDDs reduces the power consumption with 7%, because the SBDDs have 7% fewer edges on the average. Compared with [12], COMPACT based on ROBDDs and SBDDs reduces the computation delay with 46% and 52%, respectively. This stems from that crossbar designs with fewer rows are synthesized, which results in that it takes shorter time to program the memristors in the crossbar based on the Boolean input variables.

B. Comparison with other in-memory computing paradigms

In this section, we compare COMPACT with in-memory computing based on MAGIC and IMPLY logic [8], [23]. The results for [8] have been obtained from [24]. A comparison in terms of power consumption and computation delay is shown in Figure 5(a) and Figure 5(b), respectively. The figure shows that COMPACT improves the normalized computation delay from 2.15X to 3.91X. The improvements stem from that it is difficult to achieve high parallelism within the MAGIC and IMPLY logic styles. While it is possible to evaluate many "logic gates" in a single time step, the subsequent time steps will be spent attempting to realign the data to perform a highly parallel operation again. COMPACT also improves the number of devices that are required to be programmed in [8] with 1.33X. The power consumption in [8] is 1.13X due to the high number of computational steps.

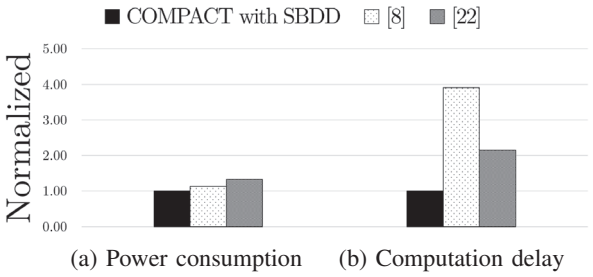


Fig. 5. Comparison of COMPACT with other logic styles for in-memory computing in terms of power and delay.

VII. Conclusion and Future Work

In this paper, we presented COMPACT for mapping Boolean functions to crossbar representations for flow-based in-memory computing. By utilizing an analogy between a BDD and a crossbar, COMPACT reduces the semiperimeter by 54% and the area by 78% compared with previous work on flow-based computing. In the future, we plan to extend COMPACT to handle limited device yield. We hypothesize that COMPACT using alternative data structures can even further improve the semiperimeter and area.

References

[1] J. Von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.
[2] J. Backus, "Can programming be liberated from the von neumann style?: A functional style and its algebra of programs," *CACM*, vol. 21, no. 8, pp. 613–641, 1978.
[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, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.
[5] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (imply) logic: Design principles and methodologies," *IEEE Transactions on VLSI Systems*, vol. 22, no. 10, pp. 2054–2066, 2013.
[6] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.
[7] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of boolean formulas using sneak paths in crossbar computing," Apr. 19 2016. US Patent 9,319,047.
[8] S. Shirinzadeh, M. Soeken, and R. Drechsler, "Multi-objective bdd optimization for rram circuit design," in *IEEE DDECS 2016*, pp. 1–6, 2016.
[9] E. Lehtonen, J. Poikonen, and M. Laiho, "Implication logic synthesis methods for memristors," in *IEEE ISCAS 2012*, pp. 2441–2444, IEEE, 2012.
[10] D. Chakraborty, S. Raj, S. L. Fernandes, and S. K. Jha, "Input-aware flow-based computing on memristor crossbars with applications to edge detection," *IEEE JETCAS 2019*, vol. 9, no. 3, pp. 580–591, 2019.
[11] A. Velasquez and S. K. Jha, "Fault-tolerant in-memory crossbar computing using quantified constraint solving," in *IEEE ICCD 2015*, pp. 101–108, IEEE, 2015.
[12] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *IEEE DATE 2017*, pp. 770–775, IEEE, 2017.
[13] A. U. Hassen, D. Chakraborty, and S. K. Jha, "Free binary decision diagram-based synthesis of compact crossbars for in-memory computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 622–626, 2018.
[14] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.
[15] S.-i. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient boolean function manipulation," in *ACM/IEEE DAC 1990*, pp. 52–57, IEEE, 1990.
[16] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *IEEE HPCA 2015*, pp. 476–488, IEEE, 2015.
[17] A. Berkeley, "A system for sequential synthesis and verification," 2009.
[18] D. B. West, *Introduction to Graph Theory*, vol. 2. Prentice hall Upper Saddle River, NJ, 1996.
[19] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, vol. 4. Springer, 2015.
[20] V. V. Vazirani, *Approximation Algorithms*. Springer Science & Business Media, 2013.
[21] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *ISMVL 2008*, pp. 220–225, IEEE, 2008.
[22] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor spice model and crossbar simulation based on devices with nanosecond switching time," in *IJCNN 2013*, pp. 1–7, IEEE, 2013.
[23] F. S. Marranghello, V. Callegaro, A. I. Reis, and R. P. Ribas, "Four-level forms for memristive material implication logic," *IEEE Transactions on VLSI Systems*, vol. 27, no. 5, pp. 1228–1232, 2019.
[24] R. B. Hur, N. Wald, N. Talati, and S. Kvatinsky, "Simple magic: Synthesis and in-memory mapping of logic execution for memristor-aided logic," in *IEEE/ACM ICCAD 2017*, pp. 225–232, IEEE, 2017.