Karnaugh Map Method for Memristive and Spintronic Asymmetric Basis Logic Functions

Vaibhav Vyas, Lucian Jiang-Wei, Peng Zhou, *Student Member, IEEE*, Xuan Hu, *Student Member, IEEE*, and Joseph S. Friedman, *Senior Member, IEEE*

Abstract—The development of beyond-CMOS technologies with alternative basis logic functions necessitates the introduction of novel design automation techniques. In particular, recently proposed computing systems based on memristors and bilayer avalanche spin-diodes both provide asymmetric functions as basis logic gates - the implication and inverted-input AND, respectively. This article therefore proposes a method by which Karnaugh maps can be directly applied to systems with asymmetric basis logic functions. A set of identities is defined for these memristor and spintronic logic functions, enabling the formal demonstration of the Karnaugh map method and an explanation of the proposed technique. This method thus, enables the direct minimization of spintronic and memristive logic circuits without translation to conventional Boolean algebra, facilitating the further development of these novel computing paradigms. Preliminary analyses demonstrate that this Karnaugh map minimization approach can provide a 28 percent reduction in step count as compared to previous manual optimization.

Index Terms—Boolean algebra, beyond-CMOS computing, asymmetric logic, emerging technologies, memristors, spintronics

1 Introduction

Emerging computing technologies provide unconventional basis logic sets that introduce new challenges for computing system design and integration. In particular, stateful logic based on memristors efficiently provides the implication and NAND functions, while a single bilayer avalanche spin-diode device can perform either an inverted-input AND (IAND) function or an OR function (see Figs. 1, 2, and 3). However, both the implication and IAND functions are non-commutative and asymmetric, inhibiting the use of conventional techniques for logic design. Many of these conventional techniques are based on Maurice Karnaugh's 1953 proposal for a map method for the minimization of Boolean logic [1]. Karnaugh's map method enables logic minimization based on the AND and OR functions, while various logic concepts based on memristors and spintronic devices provide basis logic gates for which there is no efficient technique for translation to AND and OR gates [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], preventing direct application of the Karnaugh map.

Logic minimization for stateful memristor logic has previously been investigated, but has not resulted in a minimization technique that fully exploits asymmetric logic functions. In much of the previous work, logic minimization is performed with a basis set of conventional logic functions, with each basis logic gate mapped to an efficient memristor implementation. For example, [13] minimizes a large function into NAND, OR, and parity gates, which are then realized with memristors; [14] minimizes a function into OR and inverter gates; [15] uses NOT, NAND, and OR gates; [16] uses majority gates; [17] minimizes the number of memristors (but not the step count); and [18] minimizes functions with the assistance of CMOS buffer circuits. Binary decision diagrams have also been used [19], as has an interpretation of memristors as threshold logic elements [20]. As all of these techniques minimize circuits with conventional symmetric logic functions and then implement these circuits with memristors, the circuit that is realized does not fully exploit the fundamental basis logic operations provided by memristors. Though helpful in significantly reducing circuit complexity, none of the previous works provide a technique for realizing a fully minimized circuit with asymmetric logic functions that takes into account the true physical capabilities of these devices.

A previous work in the related field of quantum computing shows the advantage of using Karnaugh maps as the starting point of logic manipulation for unusual logic functions which lead to a gate and operation count reduction compared to previous methods at the time. Similar to the asymmetric logic gates addressed in this paper, quantum circuits do not support conventional logic functions, and thus modifications of the conventional techniques are needed in order to process them natively, in an efficient manner.

The Karnaugh map method is an effective visual approach for minimizing simple logical computations, and adapting the Karnaugh map method is therefore a natural first step towards enabling efficient logic minimization for unconventional computing systems. For example, previous results with Karnaugh maps adapted for quantum circuits [21] illustrate the potential of Karnaugh maps for minimizing unusual

• *The authors are with the Department of Electrical and Computer Engineering, University of Texas at Dallas, Richardson, TX 75080 USA. E-mail: vyasvaibhav2@gmail.com, {Lucian.Jiang-Wei, peng.zhou, xuan.hu, joseph.friedman}@utdallas.edu.*

Manuscript received 26 Sept. 2019; revised 9 Feb. 2020; accepted 5 Apr. 2020. Date of publication 13 Apr. 2020; date of current version 11 Dec. 2020. (Corresponding author: Joseph S. Friedman.) Recommended for acceptance by A. L. E. Digital Object Identifier no. 10.1109/TC.2020.2986970

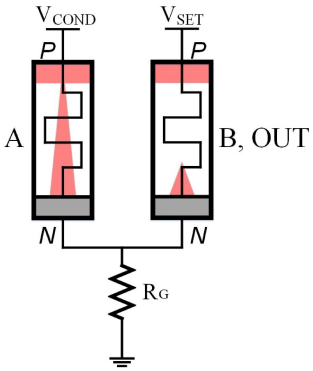


Fig. 1. Schematic of memristive implication logic, where voltages applied to the memristors modulate the resistance state.

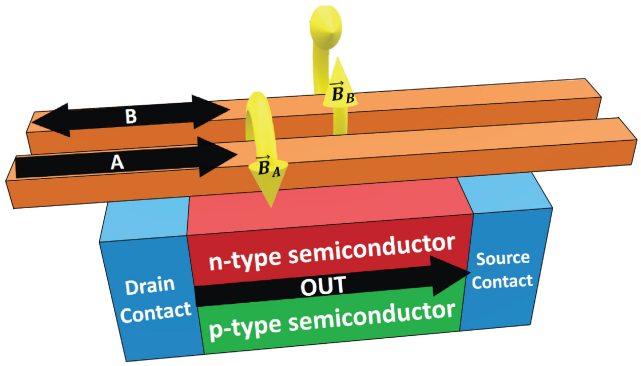


Fig. 3. Bilayer avalanche spin-diode, where the magnetic fields due to input currents A and B modulate the output current.

logic functions. This paper therefore proposes modifications of Karnaugh's map method that enable the reduction of complex asymmetric logic functions directly into minimized stateful memristor logic circuits and bilayer avalanche spin-diode logic circuits. The component devices are described in Section 2, and their asymmetric logic functionality is explained. Section 3 formally details the algebraic underpinnings of the proposed Karnaugh map method through necessary proofs of logical identities. The modified map method is proposed and explained in Section 4, and examples are provided to enable its practical use. The approach is used to algorithmically design a one-bit full adder in Section 5, demonstrating the utility of this Karnaugh map approach in the manipulation of asymmetric logic. Finally, conclusions are presented in Section 6.

2 Background

The asymmetric basis logic functions provided by stateful memristor logic and bilayer avalanche spin-diode logic can be performed compactly, but an applicable device count reduction method requires the development of techniques tailored to these functions. In particular, the implication function can be performed by two non-volatile memristors, while a NAND function can be performed by three such memristors. In bilayer avalanche spin-diode logic, a single device can perform the IAND and OR functions. The implication and IAND functions are both non-commutative and asymmetric, and their integration with NAND and OR functions, respectively, enables the development of an algebra and Karnaugh map method that enables logic minimization for both stateful memristor logic and bilayer avalanche spin-diode logic.

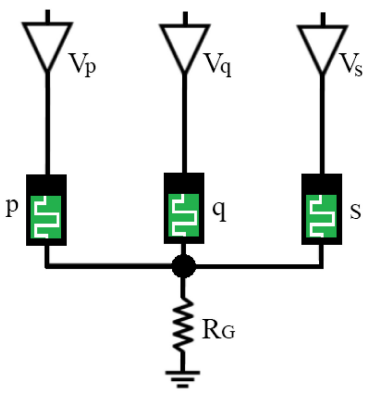


Fig. 2. Schematic for a typical NAND implementation using stateful memristive implication logic.

2.1 Stateful Memristor Logic

A memristor (or, more generally, a memristive device) is a two-terminal non-volatile device with a resistance that can be modified through application of a voltage across the two terminals [22], [23]. In general, the resistance is on a spectrum determined by the history of applied voltages; in the ideal case, applied voltages above a threshold magnitude switch a memristor between purely resistive and conductive states. The memristors shown in Fig. 1 switch to a conductive '1' state when $V_P - V_N > V_{TH}$, and a resistive '0' state when $V_N - V_P > V_{TH}$, where V_N is the voltage at node N, V_P is the voltage at node P, and V_{TH} is the threshold voltage. In some physical implementations, the resistance state is a result of the growth and retraction of a metallic filament [24].

As shown in Table 1, the implication function $OUT = A \rightarrow B$ is performed by applying V_{COND} to memristor A and V_{SET} to memristor B, where $V_{COND} < V_{TH} < V_{SET}$ and $V_{SET} - V_{COND} < V_{TH}$ [25]. When memristor A is in the resistive '0' state, the V_{SET} voltage across memristor B is greater than V_{TH}, causing memristor B to switch to the conductive '1' state or remain in that state. If memristor A is in the conductive '1' state, the voltage across memristor B is $V_{SET} - V_{COND}$; as this is less than V_{TH}, no switching occurs. This implication function (IMPLY) can thus be performed by two memristors in a single step, while a NAND function can be performed similarly with three memristors in two steps [7], [13].

Cascaded logic operations are performed in a unique manner within the stateful memristor logic paradigm: rather than each cascaded operation being performed by a distinct set of devices, the memristors are continually reused through a step-wise application of V_{SET} and V_{COND} voltages. Table 2 shows the step sequence for implementing the NAND of p and q using s as an output memristor (see Fig. 2).

TABLE 1
Truth Table for IAND and IMPLY Logic

Input A	Input B	IAND	IMPLY
0	0	0	1
0	1	0	1
1	0	1	0
1	1	0	1

TABLE 2
Computational Steps for NAND Gate With Stateful Memristive IMPLY Logic

Step	Operation	V_{COND} applied to	V_{SET} applied to	V_{RESET} applied to	Output Memristor	State of s
0	RESET	-	-	s	-	$s = 0$
1	$p \rightarrow s$	p	s	-	s	$\overline{p}$
2	$q \rightarrow s$	q	s	-	s	$\overline{p \wedge q}$

Note that while more complex operations have been proposed in a single step with ideal memristors, these operations have not been demonstrated experimentally or with physically realistic device models. This paper is therefore restricted to experimentally realizable operations with only two memristors in a single step.

2.2 Bilayer Avalanche Spin Diode Logic

In bilayer avalanche spin-diode logic, the current on two control wires modulates the current through a spin-diode [5]. These two-terminal spintronic devices have negative magnetoresistance, enabling an applied magnetic field to modulate the resistance. A constant voltage is applied across all spin-diodes at all times, thus enabling the two control wire input currents to create magnetic fields that modulate the spin-diode output current (see Fig. 3). The magnitude and direction of the magnetic fields relative to a threshold field determine the resistance state of the spin-diode. The spin-diode output currents can be used as the input control wire current to create directly cascaded logic without any amplification or control circuitry.

In this spintronic logic family, a '1' is represented by a large current while a '0' is represented by a small current. Depending upon the relative direction of current through the control wires, this spin-diode performs either the conventional OR function or the inverted-input AND (IAND) function shown in Table 1. These distinct functions result from the additive or counteractive magnetic fields created by currents oriented in the same or opposite directions, respectively. Unlike memristors, spin-diodes are not non-volatile; they return to their zero-magnetic field state immediately upon the removal of the applied input currents.

2.3 Asymmetric Basis Logic Functions

An asymmetric logic operation can be defined as one whose logic value changes when the operands are interchanged. Equivalently, these operations can be regarded as 'non-commutative' in nature. The IAND and IMPLY gates implemented using the bilayer avalanche spin-diode logic and memristors respectively, are two such asymmetric functions discussed in this paper.

As the IAND gate performs the function of an AND gate with one inverted input (Table 1), a symbol ($\barwedge$) is defined for the IAND operation such that

$$IAND(A, B) = A \wedge \overline{B} = A \barwedge B. \tag{1}$$

The symbol derives inspiration from the conventional logical operator for AND gates ($\wedge$), with the added underbar on the right arm indicating the inversion of the input to the right of the operator.

The IMPLY function is the inverse of IAND and is defined as

$$IMPLY(A, B) = A \rightarrow B = \overline{A} \vee B = \overline{A \barwedge B}. \tag{2}$$

For clarity, the input that lies on the left side (right side) of the IAND (IMPLY) operation is referred to as the *non-inverted* input, whereas the one of the right (left) is referred to as the *inverted* input. For example, A is the non-inverted input and B is the inverted input in (1); in (2), B and A are the non-inverted and inverted inputs, respectively.

Both technologies described in this paper are limited to a two-input configuration. When more than two operands are present, only two literals should be operated upon at one time. Moreover, because the operations are non-commutative, it is important to pay attention to the order of operations. The order of operations can be summarized as:

- Order of operations for IAND: *Two at a time, from left to right*

$$IAND\ (A, B, C) = (A \barwedge B) \barwedge C. \tag{3}$$

- Order of operations for IMPLY: *Two at a time, from right to left*

$$IMPLY(A, B, C) = (A \rightarrow (B \rightarrow C)). \tag{4}$$

3 Fundamental Boolean Algebra for Asymmetric Logic Functions

Before proposing the Karnaugh map method for the minimization of asymmetric logic functions, it is important to formally develop the underlying algebra. This section therefore provides a set of identities that is sufficient to demonstrate the correctness of the proposed logic minimization technique; additional identities that may be helpful for the general challenge of logic minimization will be the subject of future work. Furthermore, the identities necessary to manipulate the IAND function are stated with proofs; whereas the respective identities for IMPLY, though not proved here explicitly, can be verified using a similar methodology.

3.1 Core Algebraic Identities

As the IAND and IMPLY functions are asymmetric, modifications to conventional Boolean identities are necessary. The commutation, inversion, and association of asymmetric operations behaves quite differently, as evidenced by Theorems 1-5.

Theorem 1 (Commutative Law).

$$\text{(A) IAND}: \ A \barwedge B = \overline{B} \barwedge \overline{A} \tag{5}$$

$$\text{(B) IMPLY}: \ A \rightarrow B = \overline{B} \rightarrow \overline{A}. \tag{6}$$

Proof. Replacing the IAND with AND according to (1)

$$A \barwedge B = A \wedge \overline{B} = \overline{B} \wedge A. \tag{7}$$

Changing the RHS of (7) back to IAND notation

$$A \barwedge B = \overline{B} \barwedge \overline{A}, \tag{8}$$

which proves the theorem. □

Theorem 2 (Identity Law).

$$(\text{A})\ \text{IAND}: \ A \mathbin{\Lambda} 0 = A \tag{9}$$

$$(\text{B})\ \text{IMPLY}: \ 1 \rightarrow A = A. \tag{10}$$

Proof. Replacing IAND gates with AND gates

$$A \mathbin{\Lambda} 0 = A \wedge \overline{0} = A \wedge 1 = A. \tag{11}$$

Thus, (11) is the same as the stated theorem. □

Theorem 3 (Complement Law).

$$(\text{A})\ \text{IAND}: \ 1 \mathbin{\Lambda} A = \overline{A} \tag{12}$$

$$(\text{B})\text{IMPLY}: \ A \rightarrow 0 = \overline{A}. \tag{13}$$

Proof. Converting to conventional AND notation

$$1 \mathbin{\Lambda} A = 1 \wedge \overline{A} = \overline{A}. \tag{14}$$

Hence, the theorem is confirmed. □

Theorem 4 (Non-Inverting Associativity).

$$(\text{A})\ \text{IAND}: \ (A \mathbin{\Lambda} B) \mathbin{\Lambda} C = (A \mathbin{\Lambda} C) \mathbin{\Lambda} B \tag{15}$$

$$(\text{B})\text{IMPLY}: \ A \rightarrow (B \rightarrow C) = B \rightarrow (A \rightarrow C). \tag{16}$$

Proof. Converting the expression to AND notation

$$(A \mathbin{\Lambda} B) \mathbin{\Lambda} C = A \wedge \overline{B} \wedge \overline{C} = A \wedge \overline{C} \wedge \overline{B}. \tag{17}$$

This can be rewritten using IAND notation as follows:

$$A \wedge \overline{C} \wedge \overline{B} = (A \mathbin{\Lambda} C) \mathbin{\Lambda} B. \tag{18}$$

The theorem is thus proven. □

Theorem 5 (Inverting Associativity).

$$(\text{A})\ \text{IAND}: \ (A \mathbin{\Lambda} B) \mathbin{\Lambda} C = (\overline{C} \mathbin{\Lambda} \overline{A}) \mathbin{\Lambda} B \tag{19}$$

$$(\text{B})\text{IMPLY}: \ A \rightarrow (B \rightarrow C) = B \rightarrow (\overline{A} \rightarrow \overline{C}). \tag{20}$$

Proof. Converting to AND notation

$$(A \mathbin{\Lambda} B) \mathbin{\Lambda} C = A \wedge \overline{B} \wedge \overline{C}. \tag{21}$$

Rearranging the inputs on the RHS of the above expression

$$(A \mathbin{\Lambda} B) \mathbin{\Lambda} C = \overline{C} \wedge A \wedge \overline{B}, \tag{22}$$

which can be rewritten using IAND notation as

$$(A \mathbin{\Lambda} B) \mathbin{\Lambda} C = (\overline{C} \mathbin{\Lambda} \overline{A}) \mathbin{\Lambda} B. \tag{23}$$

Hence, the theorem is verified. □

The two associativity theorems can be applied intuitively as follows: if the non-inverted operand trades places with an inverted operand within the IAND/IMPLY expression, both of these operands are complemented to maintain logical equivalence (inverting associativity). If an inverted operand trades places with another inverted operand within the IAND/IMPLY expression, the operands are not complemented (non-inverting associativity).

3.2 Distributive Laws

In order to demonstrate the correctness of the proposed Karnaugh map method, it is helpful to present the distributive laws for IAND and IMPLY operations used in concert with OR and AND operations. As the nomenclature would be quite challenging, Theorems 6-9 are numbered rather than named.

Theorem 6 (Distributive Law - I).

$$(\text{A})\ \text{IAND}: \ A \mathbin{\Lambda} (B \wedge C) = (A \mathbin{\Lambda} B) \vee (A \mathbin{\Lambda} C). \tag{24}$$

$$(\text{B})\ \text{IMPLY}: \ A \rightarrow (B \wedge C) = (A \rightarrow B) \wedge (A \rightarrow C). \tag{25}$$

Proof. Changing LHS of (24) to AND notation and expanding using De Morgan's Law

$$A \mathbin{\Lambda} (B \wedge C) = A \wedge \overline{B \wedge C} = A \wedge (\overline{B} \vee \overline{C}). \tag{26}$$

Using the conventional OR distributive law

$$A \mathbin{\Lambda} (B \wedge C) = (A \wedge \overline{B}) \vee (A \wedge \overline{C}). \tag{27}$$

Finally, replacing the AND operations with IAND

$$A \mathbin{\Lambda} (B \wedge C) = (A \mathbin{\Lambda} B) \vee (A \mathbin{\Lambda} C), \tag{28}$$

which is the same as (24). □

Theorem 7 (Distributive Law - II).

$$(\text{A})\ \text{IAND}: \ (A \vee B) \mathbin{\Lambda} C = (A \mathbin{\Lambda} C) \vee (B \mathbin{\Lambda} C) \tag{29}$$

$$(\text{B})\text{IMPLY}: \ (A \vee B) \rightarrow C = (A \rightarrow C) \wedge (B \rightarrow C). \tag{30}$$

Proof. Changing LHS of (29) to AND notation and using the conventional OR distributive law

$$(A \vee B) \mathbin{\Lambda} C = (A \vee B) \wedge \overline{C} = (A \wedge \overline{C}) \vee (B \wedge \overline{C}). \tag{31}$$

Finally, replacing the AND operations with IAND

$$A \mathbin{\Lambda} (B \wedge C) = (A \mathbin{\Lambda} C) \vee (B \mathbin{\Lambda} C), \tag{32}$$

which is the same as (29). □

Theorem 8 (Distributive Law - III).

$$(\text{A})\ \text{IAND}: \ A \mathbin{\Lambda} (B \vee C) = (A \mathbin{\Lambda} B) \wedge (A \mathbin{\Lambda} C) \tag{33}$$

$$(\text{B})\text{IMPLY}: A \rightarrow (B \vee C) = (A \rightarrow B) \vee (A \rightarrow C). \tag{34}$$

Proof. Changing LHS of (33) to AND notation and expanding using De Morgan's Law

$$A \mathbin{\Lambda} (B \vee C) = A \wedge \overline{(B \vee C)} = A \wedge (\overline{B} \wedge \overline{C}). \tag{35}$$

This can be rewritten in IAND notation as

$$A \wedge (\overline{B} \wedge \overline{C}) = (A \wedge \overline{B}) \wedge (A \wedge \overline{C}) = (A \mathbin{\Lambda} B) \wedge (A \mathbin{\Lambda} C), \tag{36}$$

which is the same as (33). □

Theorem 9 (Distributive Law - IV).

$$\text{(A) IAND}: \; A \vee (B \barwedge C) = (A \vee B) \wedge (A \vee \overline{C}) \tag{37}$$

$$\text{(B) IMPLY}: \; A \vee (B \rightarrow C) = (\overline{A} \rightarrow \overline{B}) \vee (\overline{A} \rightarrow C) = \overline{A} \rightarrow (B \rightarrow C). \tag{38}$$

Proof. Changing the LHS of (37) to AND notation and using the conventional AND distribution law

$$A \vee (B \barwedge C) = A \vee (B \wedge \overline{C}) = (A \vee B) \wedge (A \vee \overline{C}). \tag{39}$$

The above equation validates the theorem. □

3.3 Canonical Normal Forms for Asymmetric Logic Functions

Conventional Karnaugh maps require that a function be represented as a canonical sum-of-products (SOP) or product-of-sums (POS) in which every literal is present in each term of the function; for the proposed map method for asymmetric functions, canonical sum-of-IANDs (SOI) and NAND of implications (NOI) representations are used for bilayer avalanche spin-diode and stateful memristor logic respectively.

3.3.1 Canonical Forms for IAND-OR Logic Set

The IAND and OR functions are the basis logic functions available with bilayer avalanche spin-diode logic. Similar to canonical SOP and POS expressions, canonical SOI expressions can be developed from non-canonical SOI expressions by adding the literals missing from each term. Applying Theorem 2, appending an IAND operation of a null-valued (zero) expression (such as $A \wedge \overline{A}$) enables the expansion without modifying the logical value of the expression. Taking the non-canonical SOI expression

$$f_{soi} = ((\overline{B} \barwedge \overline{C}) \barwedge \overline{D}) \vee ((A \barwedge B) \barwedge \overline{C}). \tag{40}$$

$A \wedge \overline{A}$ and $D \wedge \overline{D}$ can be appended to the two terms, resulting in

$$f_{soi} = \{((\overline{B} \barwedge \overline{C}) \barwedge \overline{D}) \barwedge (A \wedge \overline{A})\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge (D \wedge \overline{D})\}. \tag{41}$$

Using Theorem 6 (A)

$$f_{soi} = \{((\overline{B} \barwedge \overline{C}) \barwedge \overline{D}) \barwedge A\} \vee \{((\overline{B} \barwedge \overline{C}) \barwedge \overline{D}) \barwedge \overline{A}\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge D\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}\}. \tag{42}$$

Rearranging the inverted and non-inverted inputs according to Theorem 5 (A)

$$f_{soi} = \{((\overline{A} \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge D\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}\}. \tag{43}$$

Finally, by conventional OR idempotency, $A \vee A = A$

$$f_{soi} = \{((\overline{A} \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge D\} \vee \{((A \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}\}. \tag{44}$$

The above expression is a canonical SOI expression. In general, canonical IAND-of-sums (IOS) expressions can be achieved with the same method as canonical POS: by OR-ing a null expression of the missing literals with each of the sum-terms, followed by ordinary Boolean algebraic simplification.

3.3.2 Canonical Forms for IMPLY-NAND Logic Set

A given Boolean expression may not be in its canonical form, and thus similar to the methodology adopted for IANDs, canonical NOI expressions can be developed from non-canonical NOI expressions by inserting the literals missing from each term while maintaining logical equivalence. Stateful memristor logic provides IMPLY and NAND operations as the basis logic set. Performing an IMPLY operation of a unity '1'-valued expression (such as $A \vee \overline{A}$), followed by Boolean reduction as per the identities in the previous section, expands the function into a canonical form while retaining its logical value as per Theorem 2 (B). Consider the non-canonical NOI expression in (45)

$$f_{noi} = \overline{(\overline{B} \rightarrow (\overline{C} \rightarrow \overline{D})) \wedge (A \rightarrow (B \rightarrow \overline{C}))}. \tag{45}$$

$A \vee \overline{A}$ and $D \vee \overline{D}$ can be appended to the two terms, resulting in

$$f_{noi} = \overline{\{(A \vee \overline{A}) \rightarrow (\overline{B} \rightarrow (\overline{C} \rightarrow \overline{D}))\}} \; \overline{\wedge \{(D \vee \overline{D}) \rightarrow (A \rightarrow (B \rightarrow \overline{C}))\}}. \tag{46}$$

Using Theorem 7 (B)

$$f_{noi} = \overline{\{A \rightarrow (\overline{B} \rightarrow (\overline{C} \rightarrow \overline{D}))\}} \; \overline{\wedge \{\overline{A} \rightarrow (\overline{B} \rightarrow (\overline{C} \rightarrow \overline{D}))\} \wedge \{D \rightarrow (A \rightarrow (B \rightarrow \overline{C}))\}} \; \overline{\wedge \{\overline{D} \rightarrow (A \rightarrow (B \rightarrow \overline{C}))\}}. \tag{47}$$

Rearranging the inverted and non-inverted inputs according to Theorems 4(B) and 5(B)

$$f_{noi} = \overline{\{A \rightarrow (\overline{B} \rightarrow (\overline{C} \rightarrow \overline{D}))\}} \; \overline{\wedge \{\overline{A} \rightarrow (\overline{B} \rightarrow (\overline{C} \rightarrow \overline{D}))\} \wedge \{A \rightarrow (B \rightarrow (C \rightarrow \overline{D}))\}} \; \overline{\wedge \{A \rightarrow (B \rightarrow (C \rightarrow D))\}}. \tag{48}$$

The above expression is a canonical NOI expression. In general, canonical Implication-of-NANDs (ION) expressions can be achieved with the same method as canonical POS: by AND-ing a unity expression of the missing literals with each of the sum-terms, followed by ordinary Boolean algebraic simplification.

4 Karnaugh Map Method for Asymmetric Logic Functions

The proposed Karnaugh map method enables a graphical technique for the minimization of memristor and spintronic logic with asymmetric basis functions. Following the explanation of conventional Karnaugh maps below, the adapted Karnaugh map method is described and its operation is explained. The translation between stateful memristor logic and bilayer avalanche spin-diode logic is shown, followed by examples that provide instruction as to the use of the proposed method.

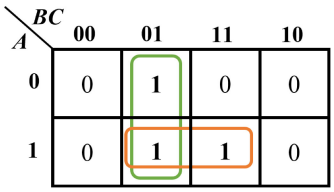


Fig. 4. Karnaugh map for the example in Section 4.1 (49) and the two examples in Section 4.2 (53) and (60).

4.1 Background: Conventional Karnaugh Maps

Karnaugh Maps are tabular representations of Boolean logic functions consisting of 2^n cells, each for one among the possible combinations of n-bit binary data. The cells are arranged such that logically adjacent terms share physical adjacency. Graphical pairing of these adjacent terms reduces the function to its essential prime implicants [1], [26]. Whereas Karnaugh originally described the method only through examples [1], this paper endeavors to formally demonstrate the validity of the proposed method.

A conventional Karnaugh map is shown in Fig. 4 for the expression

$$f_{sop} = (\overline{A} \wedge \overline{B} \wedge C) \vee (A \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge C). \tag{49}$$

This expression can be rewritten by duplicating the term $A \wedge \overline{B} \wedge C$

$$f_{sop} = \{(\overline{A} \wedge \overline{B} \wedge C) \vee (A \wedge \overline{B} \wedge C)\} \vee \{(A \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge C)\}. \tag{50}$$

This expression can then be simplified according to conventional Boolean algebra techniques, first with the OR distributive law

$$f_{sop} = ((A \vee \overline{A}) \wedge \overline{B} \wedge C) \vee ((B \vee \overline{B}) \wedge A \wedge C). \tag{51}$$

$A \vee \overline{A}$ and $B \vee \overline{B}$ are '1' by complement law, enabling the elimination of operands A and B from the respective product terms

$$f_{sop} = (1 \wedge \overline{B} \wedge C) \vee (1 \wedge A \wedge C) = (\overline{B} \wedge C) \vee (A \wedge C). \tag{52}$$

Karnaugh's method reaches this result in a similar manner, but graphically. This can be noted in the Karnaugh map pair encircled with green and orange (Fig. 4): The logical adjacency enables the combination of literals with their complements and therefore minimize redundancies in a function.

4.2 Map Method for Asymmetric Functions

The proposed map method for asymmetric logic functions is performed in the following step-wise manner:

1. Transform the target expression into its canonical form (i.e., SOI/IOS or NOI/ION).
2. Mark the corresponding SOI/NOI terms in the cells of a Karnaugh map.
3. Group the minterms/maxterms graphically according to the standard Karnaugh map techniques described in [1].
4. Use the standard rules of the Karnaugh map to create the expression with the resultant terms.
5. Unless the left-most (right-most) operand is equal to the left-most (right-most) operand in an IAND (IMPLY) equation's canonical form, it should be complemented.

To illustrate this method, the equation displayed in the K-map in Fig. 4 will be reused. In this four-bit function, the order of the variables are $\{A, B, C, D\}$, meaning that any of the variables $\{B, C, D\}$ must be inverted if they are the left-most operand in the simplified equation. The following systematic evaluation of the equation will highlight the necessity of the fifth step

$$f_{soi} = ((\overline{A} \,\Lambda\, \overline{B}) \,\Lambda\, C) \vee ((A \,\Lambda\, \overline{B}) \,\Lambda\, C) \vee ((A \,\Lambda\, B) \,\Lambda\, C). \tag{53}$$

This expression can be simplified by applying the theorems outlined in Section 3.1. First using Theorem 7(A) twice in succession gives (54) and (55)

$$f_{soi} = (((\overline{A} \,\Lambda\, \overline{B}) \vee (A \,\Lambda\, \overline{B})) \,\Lambda\, C) \vee ((A \,\Lambda\, B) \,\Lambda\, C) \tag{54}$$

$$f_{soi} = (((\overline{A} \vee A) \,\Lambda\, \overline{B}) \,\Lambda\, C) \vee ((A \,\Lambda\, B) \,\Lambda\, C). \tag{55}$$

$\overline{A} \vee A$ is unity due to the OR complement law, and $1 \,\Lambda\, \overline{B} = B$ according to Theorem 3(A). Thus

$$f_{soi} = ((1 \,\Lambda\, \overline{B}) \,\Lambda\, C) \vee ((A \,\Lambda\, B) \,\Lambda\, C) = (B \,\Lambda\, C) \vee ((A \,\Lambda\, B) \,\Lambda\, C). \tag{56}$$

Due to the nature of the complement law for IAND, reading the same K-map will yield different results even when the pairings stay constant. For conventional logic, $1 \wedge \overline{B} = \overline{B}$, although for the IAND, $1 \,\Lambda\, \overline{B} = B$. Due to this discrepancy, it is necessary to complement B in the final equation, in order to maintain logical equivalence. The expression can be further reduced by applying Theorems 6(A) and 9(A)

$$f_{soi} = (B \vee (A \,\Lambda\, B)) \,\Lambda\, C \tag{57}$$

$$f_{soi} = ((B \vee A) \wedge (B \vee \overline{B})) \,\Lambda\, C. \tag{58}$$

With the conventional OR complement law, and $B \vee \overline{B} = 1$ using Theorem 7(A), this becomes

$$f_{soi} = (B \,\Lambda\, C) \vee (A \,\Lambda\, C). \tag{59}$$

The same result is obtained by applying the proposed methodology to the Karnaugh map in Fig. 4.

Now consider the SOI expression in (53) represented as a canonical NOI in (60)

$$f_{noi} = \overline{(\overline{A} \to (\overline{B} \to C)) \wedge (A \to (\overline{B} \to C))} \wedge \overline{(A \to (B \to C))}. \tag{60}$$

Similar to SOI, the above expression can be reduced using the proposed Boolean laws for IMPLY logic (Section 3)

$$f_{noi} = \overline{(\overline{B} \to C) \wedge (A \to C)}. \tag{61}$$

Expression (61) shows the minimized form of (60). Now, applying the modified Karnaugh method proposed in this section to the expression in (60) (Karnaugh map shown in Fig. 4), we obtain

$$f_{noi} = \overline{(\overline{B} \to C) \wedge (A \to C)}. \tag{62}$$

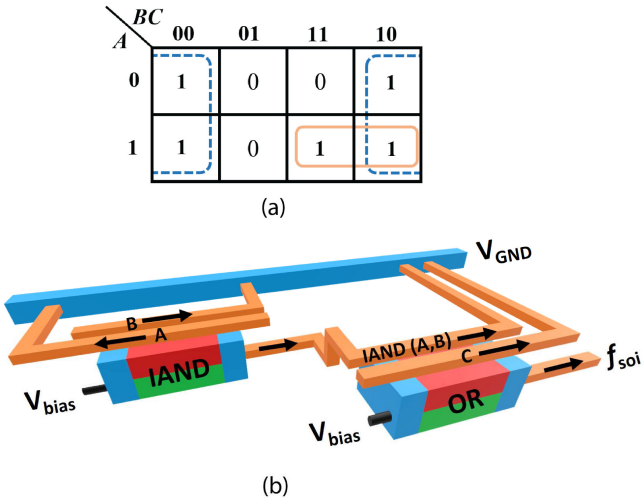


Fig. 5. (a) Karnaugh map and (b) minimized physical circuit for Example 4.3.

Note that complementation is not required in this instance, as none of the inverted inputs appear as the left-most operand in the reduced equation. Hence, (61) and (62) illustrate that the proposed method is valid for NOI as well.

4.3 Examples

To clearly explain the mapping methodology, we demonstrate the application of Karnaugh maps to the decomposition of a few sample Boolean expressions involving IAND-OR and IMPLY-NAND logic sets. The result for each example can be validated by converting and solving them as SOP. Possible circuit implementations have also been shown for each example.

Example 1. (Three-Variable SOI). Consider the following SOI:

$$f_{soi} = \{((\overline{A} \wedge \overline{B}) \wedge \overline{C}) \vee ((\overline{A} \wedge B) \wedge \overline{C}) \vee ((A \wedge \overline{B}) \wedge \overline{C}) \vee ((A \wedge B) \wedge \overline{C}) \vee ((A \wedge B) \wedge C)\}. \tag{63}$$

Converting to conventional SOP

$$f_{sop} = \{(\overline{A} \wedge B \wedge C) \vee (\overline{A} \wedge \overline{B} \wedge C) \vee (A \wedge B \wedge C) \vee (A \wedge \overline{B} \wedge C) \vee (A \wedge \overline{B} \wedge \overline{C})\}. \tag{64}$$

Grouping the terms and applying the conventional OR complement law successively

$$f_{sop} = C \vee (A \wedge \overline{B} \wedge \overline{C}). \tag{65}$$

Using the AND distributive law and OR Complement Law in succession

$$f_{sop} = (C \vee A \wedge \overline{B}) \wedge (C \vee \overline{C}) \tag{66}$$

$$f_{sop} = C \vee (A \wedge \overline{B}), \tag{67}$$

which can be rewritten as SOI

$$f_{soi}(A, B, C) = C \vee (A \barwedge B). \tag{68}$$

The same result is obtained by interpreting the Karnaugh map in Fig. 5a, hence validating the proposed method. Fig. 5b shows the device-level implementation for (68).

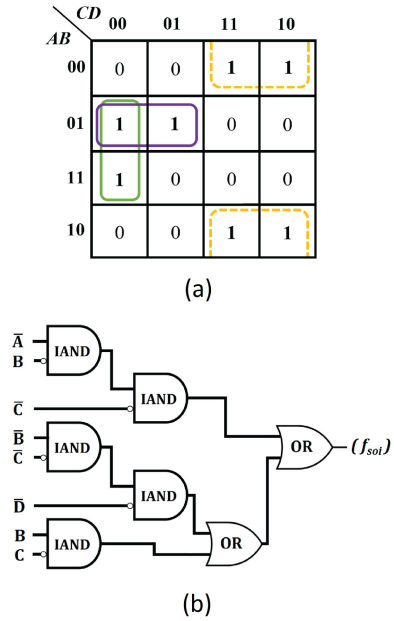


Fig. 6. (a) Karnaugh map and (b) minimized schematic for Example 4.3.

Example 2. (Four-Variable SOI). The following SOI expression is plotted in the Karnaugh map of Fig. 6a

$$f_{soi} = (((\overline{A} \barwedge \overline{B}) \barwedge C) \barwedge \overline{D}) \vee (((\overline{A} \barwedge \overline{B}) \barwedge C) \barwedge D) \vee (((\overline{A} \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}) \vee (((\overline{A} \barwedge B) \barwedge \overline{C}) \barwedge D) \vee (((A \barwedge \overline{B}) \barwedge C) \barwedge \overline{D}) \vee (((A \barwedge \overline{B}) \barwedge C) \barwedge D) \vee (((A \barwedge B) \barwedge \overline{C}) \barwedge \overline{D}). \tag{69}$$

The corresponding simplified function deduced by applying the mapping method outlined in Section 4.2 is given by (70). Note that literal B in the second and third IAND terms has been inverted

$$f_{soi} = ((\overline{A} \barwedge B) \barwedge \overline{C}) \vee ((\overline{B} \barwedge \overline{C}) \barwedge \overline{D}) \vee (B \barwedge C). \tag{70}$$

Gate-level circuit implementation using IAND and OR gates is shown in Fig. 6b.

Example 3. (Three-Variable NOI). This example illustrates the optimization of an NOI expression and its implementation using the IMPLY-NAND logic set (Fig. 7). For analogy, consider the function in (63), now represented as an NOI (note that (63) and (71) are not equivalent) and plotted on a Karnaugh map as shown in Fig. 7a

$$f_{noi} = \overline{\{(A \rightarrow (\overline{B} \rightarrow C)) \wedge (A \rightarrow (B \rightarrow C)) \wedge (\overline{A} \rightarrow (\overline{B} \rightarrow C)) \wedge (\overline{A} \rightarrow (B \rightarrow C)) \wedge (\overline{A} \rightarrow (B \rightarrow \overline{C}))\}}. \tag{71}$$

Using the proposed mapping method, non-inverted input B is complemented and the simplified function is written as

$$f_{noi} = \overline{\overline{C} \wedge (\overline{A} \rightarrow \overline{B})}. \tag{72}$$

Fig. 7b shows a possible memristor circuit for the IMPLY-NAND implementation. Here, complementary representation [13] is used and the function can be reached

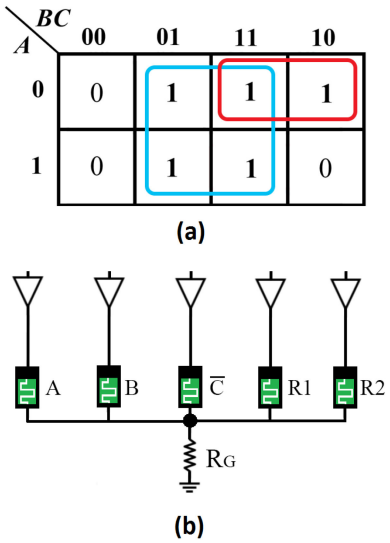


Fig. 7. Example 4.3: (a) Karnaugh map for (71). (b) Memristor circuit for IMPLY-NAND implementation of (72), where memristors A, B, and $\overline{C}$ contain the input values, while R_1 and R_2 are output memristors.

step-wise by executing the computational sequence listed in Table 3 [13].

Example 4. (Incompletely-Specified Four-Variable NOI). Expression (73) shows a Boolean function represented as an NOI with a "don't care" term. Fig. 8a shows the Karnaugh map and reduced groupings of

$$f_{noi} = \overline{\{(\overline{A} \rightarrow (B \rightarrow (\overline{C} \rightarrow \overline{D}))) \wedge (\overline{A} \rightarrow (B \rightarrow (\overline{C} \rightarrow D)))} \overline{\wedge (A \rightarrow (B \rightarrow (\overline{C} \rightarrow \overline{D}))) \wedge (A \rightarrow (B \rightarrow (\overline{C} \rightarrow D)))} \overline{\wedge (\overline{A} \rightarrow (\overline{B} \rightarrow (C \rightarrow \overline{D}))) \wedge (\overline{A} \rightarrow (\overline{B} \rightarrow (C \rightarrow D)))^{d}\}}. \tag{73}$$

The above expression is reduced to (74) by following the proposed mapping process. In this case, the literal C is complemented in both of the terms in accordance with the steps of the proposed map method. Therefore, the minimized function is

$$f_{noi} = \overline{(B \rightarrow C) \wedge (\overline{A} \rightarrow (\overline{B} \rightarrow \overline{C}))}. \tag{74}$$

Fig. 8b shows a memristive circuit with complementary representation for the implementation of (74). Table 4 shows the minimized sequence of operations that leads to the desired result being stored in the memristor R_2.

TABLE 3
Computational Steps for Implementation of (72) Using Stateful Memristive IMPLY Logic

Step	Operation	V_{COND} applied to	V_{SET} applied to	V_{RESET} applied to	Output Memristor	Result
0	RESET	-	-	R_1, R_2	-	$R_1 = R_2 = 0$
1	$B \rightarrow R_2$	B	R_2	-	R_2	$\overline{B}$
2	$A \rightarrow R_2$	A	R_2	-	R_2	$A \rightarrow \overline{B}$
3	$R_2 \rightarrow R_1$	R_2	R_1	-	R_1	$(A \rightarrow \overline{B})$
4	$\overline{C} \rightarrow R_2$	$\overline{C}$	R_2	-	R_2	Expression (72)

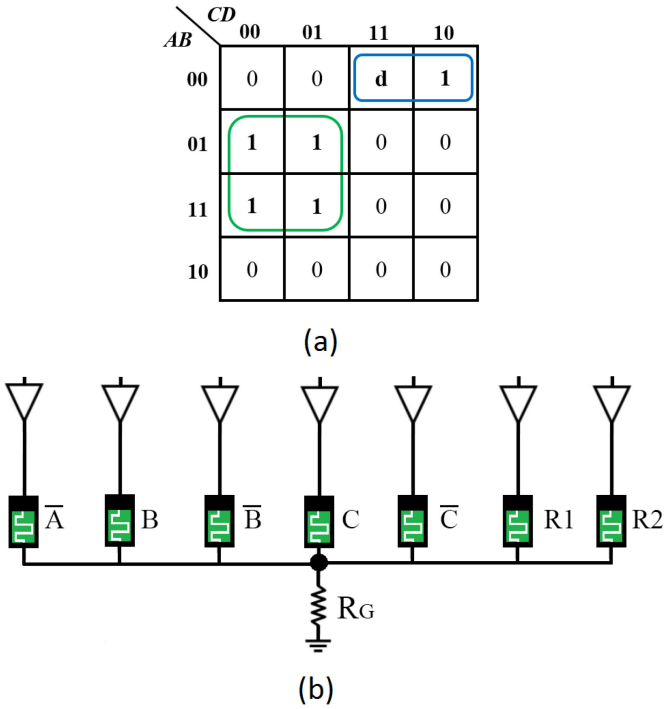


Fig. 8. (a) Karnaugh map for (73). (b) Memristor circuit for IMPLY-NAND implementation of (74), where memristors A, B, $\overline{B}$, C, and $\overline{C}$ contain the input values, while R_1 and R_2 are output memristors.

5 Algorithmically-Designed Memristor Full Adder

To demonstrate its utility, this novel Karnaugh map approach is applied to the memristor full adder composed of IMPLY and NAND gates. The equations for the sum and carry-out of the full adder are reduced separately, with each having their own Karnaugh map. This Karnaugh map method algorithmically designs a full adder circuit that reduces the required number of steps by 28 percent.

The Karnaugh maps for the sum and carry-out are first determined from Table 5, with the sum equation resulting in (75) and the carry out equation resulting in (76).

$$f_{noi} = \overline{\{(A \rightarrow (\overline{B} \rightarrow \overline{C})) \wedge (\overline{A} \rightarrow (\overline{B} \rightarrow C))} \overline{\wedge (A \rightarrow (B \rightarrow C)) \wedge \ (\overline{A} \rightarrow (B \rightarrow \overline{C}))\}} \tag{75}$$

TABLE 4
Computational Steps for Implementation of (74) Using Stateful Memristive IMPLY Logic

Step	Operation	V_{COND} applied to	V_{SET} applied to	V_{RESET} applied to	Output Memristor	Result
0	RESET	-	-	R_1, R_2	-	$R_1 = R_2 = 0$
1	$\overline{C} \rightarrow R_2$	$\overline{C}$	R_2	-	R_2	C
2	$B \rightarrow R_2$	B	R_2	-	R_2	$\overline{B \rightarrow C}$
3	$R_2 \rightarrow R_1$	R_2	R_1	-	R_1	$\overline{(B \rightarrow C)}$
4	$R_2 = 0$	-	-	R_2	-	-
5	$C \rightarrow R_2$	C	R_2	-	R_2	$\overline{C}$
6	$\overline{B} \rightarrow R_2$	$\overline{B}$	R_2	-	R_2	$\overline{B} \rightarrow \overline{C}$
7	$\overline{A} \rightarrow R_2$	$\overline{A}$	R_2	-	R_2	$\overline{A} \rightarrow (\overline{B} \rightarrow \overline{C})$
8	$R_2 \rightarrow R_1$	R_2	R_1	-	R_1	Expression (74)

TABLE 5
Truth Table for a Full Adder

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$f_{noi} = \overline{\{(\overline{A} \rightarrow (B \rightarrow C)) \wedge (A \rightarrow (\overline{B} \rightarrow C)) \wedge (A \rightarrow (B \rightarrow \overline{C})) \wedge \ (A \rightarrow (B \rightarrow C))\}}. \quad (76)$$

The two equations are mapped using the method described in [1], with the resulting K-maps shown in Fig. 9. As the Karnaugh map for the sum of the full adder contains only singletons, no further reduction is possible. The equation for the IMPLY-NAND full adder sum can be directly derived from the Karnaugh map, using the methods discussed in Section 4.2. In the case of the carry-out, some coverage can be achieved using the traditional Karnaugh map grouping techniques from [1], allowing further reduction of the equation. The reduced equation may be derived in NOI form from the Karnaugh map, following the steps in Section 4.2. The reduced carry-out equation is

$$f_{noi} = \overline{(A \rightarrow \overline{C}) \wedge (B \rightarrow \overline{C}) \wedge (A \rightarrow \overline{B})}. \quad (77)$$

Note that the second operand in each term has been inverted, as none of those operands are equal to the right-most operand in the IMPLY equation's canonical form.

The resulting fully-reduced logical step-wise procedure is depicted in Fig. 10. This algorithmically-designed full adder requires seven steps for the carry-out computation and 14 steps for the sum computation, for a total of 21 total steps. This is a 28 percent reduction from the state-of-the-art [7], which was optimized manually. As manual approaches generally outperform algorithmic procedures for small-scale circuit optimization, this significant improvement is quite remarkable. Furthermore, it should be noted that while the algorithmically-designed full adder requires a larger device

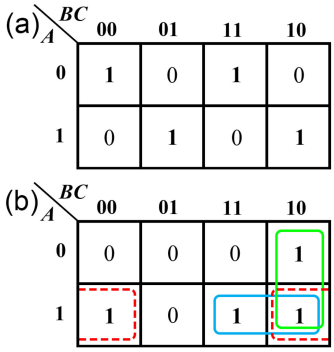


Fig. 9. Karnaugh maps for the full adder equations. Values have been mapped directly from the truth table. (a) Karnaugh map for (75). (b) Karnaugh map for (76).

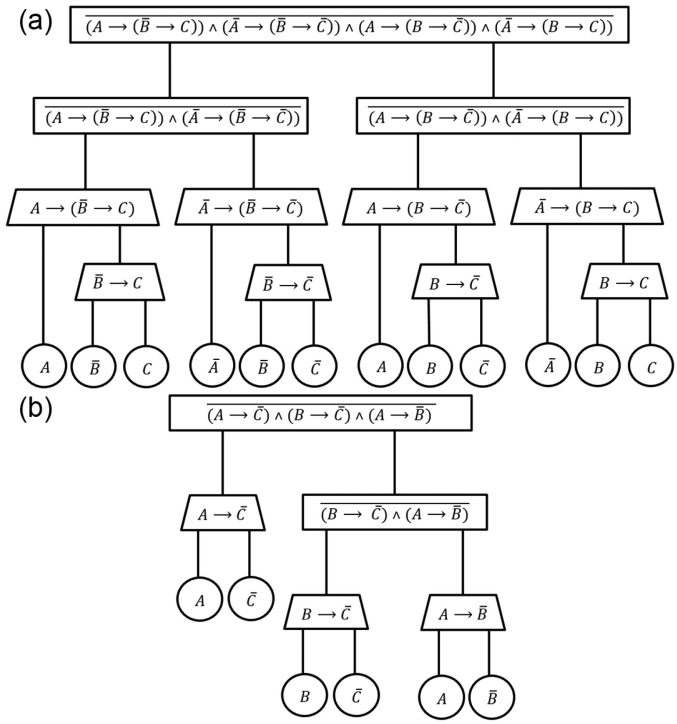


Fig. 10. Logical abstractions for each equation. Input variables are designated with circles, IMPLY operations are designated with trapezoids, and NAND operations are designated with rectangles. (a) Logical abstraction for (75). (b) Logical abstraction for the reduced C_{out} Equation (77).

count, the number of steps is far more important when considering the efficiency of the complete stateful memristor logic system [27].

The 28 percent reduction in step count is specific to the one-bit full adder, but can be scaled up to multi-bit full adders. Therefore, for an N-bit full adder, the K-map-optimized circuit requires $14N$ and $7N$ steps to implement Sum and C_{out}, respectively, retaining the 28 percent step reduction for multi-bit adders. Furthermore, if Sum and C_{out} are computed in parallel, Sum requires $7N + 7$ steps while C_{out} requires $7N$ steps, boosting the overall step reduction to 76 percent. The proposed method thus has the potential to achieve step reductions greater than 28 percent when applied to large systems.

6 Conclusion

The logic minimization method proposed here enables the direct mapping of memristive and spintronic logic functions onto Karnaugh maps, which has been shown in the past to be a valuable first step in creating logic manipulation techniques for novel devices with unconventional logic functions. This method is tailored to the asymmetric IMPLY and IAND logic functions, and the NAND and OR functions that are also efficiently performed by memristors and bilayer avalanche spin-diodes, respectively. The identities defined here enable algebraic manipulation of these functions, which is used to formally demonstrate the validity of the proposed logic minimization technique. This logic minimization technique provides a foundation for logic reduction based on memristors and spintronic logic, as well as a template for logic minimization with alternative beyond-CMOS computing structures. Furthermore, algorithmic design using this Karnaugh map

method has been shown to provide a 28 percent reduction in step count as compared to the best manually-minimized full adder circuits. This Karnaugh map method adapted to non-commutative logic functions thus constitutes an important step toward the development of logic minimization techniques for the next generation of computing.

REFERENCES

[1] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Trans. Amer. Inst. Elect. Engineers, Part I: Commun. Electron.*, vol. 72, no. 5, pp. 593–599, Nov. 1953.

[2] M. T. Niemier *et al.*, "Nanomagnet logic: Progress toward system-level integration," *J. Phys., Condens. Matter*, vol. 23, no. 49, 2011, Art. no. 493202.

[3] D. Hampel and R. O. Winder, "Threshold logic," *IEEE Spectr.*, vol. 8, no. 5, pp. 32–39, May 1971.

[4] A. Imre,"Majority logic gate for magnetic quantum-dot cellular automata," *Science*, vol. 311, no. 5758, pp. 205–208, 2006.

[5] J. S. Friedman, E. R. Fadel, B. W. Wessels, D. Querlioz, and A. V. Sahakian, "Bilayer avalanche spin-diode logic," *AIP Advances*, vol. 5, no. 11, 2015, Art. no. 117102.

[6] J. S. Friedman *et al.*, "Cascaded spintronic logic with low-dimensional carbon," *Nat. Commun.*, vol. 8, 2017, Art. no. 15635.

[7] 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 Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, Oct. 2014.

[8] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, 2009, pp. 33–36.

[9] D. E. Nikonov, G. I. Bourianoff, and T. Ghani, "Proposal of a spin torque majority gate logic," *IEEE Electron Device Lett.*, vol. 32, no. 8, pp. 1128–1130, Aug. 2011.

[10] M. H. Ben-Jamaa, K. Mohanram, and G. De Micheli, "An efficient gate library for ambipolar CNTFET logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 242–255, Feb. 2011.

[11] H. Zhang, W. Kang, L. Wang, K. L. Wang, and W. Zhao, "Stateful reconfigurable logic via a single-voltage-gated spin hall-effect driven magnetic tunnel junction in a spintronic memory," *IEEE Trans. Electron Devices*, vol. 64, no. 10, pp. 4295–4301, Oct. 2017.

[12] H. Zhang, W. Kang, K. Cao, B. Wu, Y. Zhang, and W. Zhao, "Spintronic processing unit in spin transfer torque magnetic random access memory," *IEEE Trans. Electron Devices*, vol. 66, no. 4, pp. 2017–2022, Apr. 2019.

[13] E. Lehtonen, J. Poikonen, and M. Laiho, "Implication logic synthesis methods for memristors," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2012, pp. 2441–2444.

[14] A. Chattopadhyay and Z. Rakosi, "Combinational logic synthesis for material implication," in *Proc. IEEE/IFIP 19th Int. Conf. VLSI Syst.-on-Chip*, 2011, pp. 200–203.

[15] A. Raghuvanshi and M. Perkowski, "Logic synthesis and a generalized notation for memristor-realized material implication gates," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2014, pp. 470–477.

[16] S. Shirinzadeh, M. Soeken, and R. Drechsler, "Multi-objective BDD optimization for RRAM based circuit design," in *Proc. IEEE 19th Int. Symp. Des. Diagnostics Electron. Circuits Syst.*, 2016, pp. 1–6.

[17] F. S. Marranghello, V. Callegaro, A. I. Reis, and R. P. Ribas, "SOP based logic synthesis for memristive IMPLY stateful logic," in *Proc. 33rd IEEE Int. Conf. Comput. Des.*, 2015, pp. 228–235.

[18] F. Lalchhandama, B. G. Sapui, and K. Datta, "An improved approach for the synthesis of Boolean functions using memristor based IMPLY and INVERSE-IMPLY gates," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2016, pp. 319–324.

[19] S. Chakraborti, P. V. Chowdhary, K. Datta, and I. Sengupta, "BDD based synthesis of boolean functions using memristors," in *Proc. 9th Int. Des. Test Symp.*, 2014, pp. 136–141.

[20] D. Fan, M. Sharad, and K. Roy, "Design and synthesis of ultralow energy spin-memristor threshold logic," *IEEE Trans. Nanotechnol.*, vol. 13, no. 3, pp. 574–583, May 2014.

[21] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "Modified karnaugh map for quantum Boolean circuits construction," in *Proc. 3rd IEEE Conf. Nanotechnol.*, 2003, pp. 651–654.

[22] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep. 1971.

[23] D. B. Strukov, G. S. Snider, D. R. Stewart, andR. S. Williams,"The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

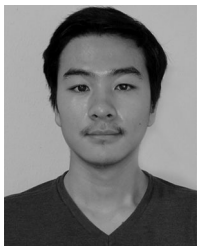
[24] Y. Yang, P. Gao, S. Gaba, T. Chang, X. Pan, and W. Lu, "Observation of conducting filament growth in nanoscale resistive memories," *Nat. Commun.*, vol. 3, 2012, Art. no. 732.

[25] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams,"'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

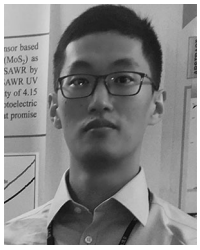
[26] J. F. Miller, "Principles in the evolutionary design of digital circuits - Part I," *Genetic Program. Evolvable Mach.*, vol. 1, pp. 7–35, 2000.

[27] X. Hu, M. J. Schultis, M. Kramer, A. Bagla, A. Shetty, and J. S. Friedman, "Overhead requirements for stateful memristor logic," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 263–273, Jan. 2019.

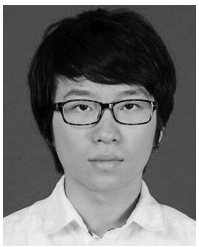
Vaibhav Vyas received the BE degree in electronics and communication engineering from the Rajiv Gandhi Proudyogiki Vishwavidyalaya (R.G.P.V.) University, Bhopal, Madhya Pradesh, India, in 2016, and the MS degree in electrical engineering from the Erik Jonsson School of Engineering & Computer Science, University of Texas at Dallas, Richardson, Texas, in 2018. He continues to do research on circuit design and logic design automation for bilayer avalanche spin-diode logic and stateful memristor logic. After graduating from UT Dallas, he was working with John Deere Intelligent Solutions Group, Des Moines (IA) as an embedded software engineer with the Precision Ag Department. In the September of 2019, he joined the Firmware Engineering Team, Extron Electronics, working on a variety of Extron products.



Lucian Jiang-Wei received the BS degree in computer science from the Erik Jonsson School of Engineering & Computer Science, University of Texas at Dallas, Richardson, Texas, in 2019. He continues to do research on logic design automation for spintronic and emerging technologies.



Peng Zhou (Student Member, IEEE) received the bachelor's degree in mechanical engineering from the Xiamen University of Technology, Xiamen, China, in 2015, and the master's degree in electrical engineering from Xiamen University, Xiamen, China, in 2018. He is working toward the PhD degree in electrical engineering in the Erik Jonsson School of Engineering & Computer Science, University of Texas at Dallas, Richardson, Texas. His current research focus is on neurmorphic computing with STT-MRAM.



Xuan Hu (Student Member, IEEE) received the BS degree in electrical and information engineering from Huaqiao University, Xiamen, China, in 2013, and the MS degree in electrical engineering from Arizona State University, Tempe, Arizona, in 2015. He is currently working toward the PhD degree in electrical engineering in the Erik Jonsson School of Engineering & Computer Science, University of Texas at Dallas, Richardson, Texas. His current research focus is on circuit design and modeling of efficient memristive, spintronic, and carbon nanotube logic circuits.

Joseph S. Friedman (Senior Member, IEEE) received the AB and BE degrees from Dartmouth College, Hanover, New Hampshire, in 2009, and the MS and PhD degrees in electrical & computer engineering from Northwestern University, Evanston, Illinois, in 2010 and 2014, respectively. He joined the University of Texas at Dallas, Richardson, Texas, in 2016, where he is currently an assistant professor of electrical & computer engineering and the director of the NeuroSpinCompute Laboratory. From 2014 to 2016, he was a Centre National de la Recherche Scientifique research associate with the Institut d'Electronique Fondamentale, Université Paris-Sud, Orsay, France. He has also been a summer faculty fellow with the U.S. Air Force Research Laboratory, Rome, New York, a visiting professor with Politecnico di Torino, Turin, Italy, a guest scientist with RWTH Aachen University, Aachen, Germany, and worked on logic design automation as an intern with Intel Corporation, Santa Clara, California. He is a member of the editorial board of the *Microelectronics Journal*, the technical program committees of DAC, DATE, SPIE Spintronics, NANOARCH, GLSVSI, ICECS, VLSI-SoC, the review committee of ISCAS, and the IEEE Circuits & Systems Society Nanoelectronics and Gigascale Systems Technical Committee. He has been a member of the organizing committee of VLSI-SoC 2020, NANOARCH 2019, and DCAS 2018. He has also been awarded a Fulbright Postdoctoral Fellowship. His research interests include the invention and design of novel logical and neuromorphic computing paradigms based on nanoscale and quantum mechanical phenomena, with particular emphasis on spintronics.