A Semiparallel Full-Adder in IMPLY Logic

Shokat Ganjeheizadeh Rohani, Nima Taherinejad, and David Radakovits

***Abstract*— Passive implementation of memristors has led to several innovative works in the field of electronics. Despite being primarily a candidate for memory applications, memristors have proven to be beneficial in several other circuits and applications as well. One of the use cases is the implementation of digital circuits such as adders. Among several logic implementations using memristors, IMPLY logic is one of the promising candidates. In this brief, we present a new architecture for a digital full-adder, which is up to 41% faster than existing IMPLY-based serial designs while requiring up to 78% less area (memristors) compared to the existing parallel design.**

***Index Terms*— full-adder, IMPLY, in-memory computation, material implication, memristive, memristor, ReRAM, semi-parallel, stateful logic.**

I. Introduction

A natural application of memristors is their use in memory systems [1]–[5]. Application of memristors has been explored in several other fields too. Learning [6]–[8], digital circuits [8]–[10], quantum computing [8], and cancer detection [11] are some of those applications. In this brief, we focus on the use of memristors in digital circuits. In particular, full-adder is one of the fundamental blocks of many computing systems. There have been several efforts made in devising memristor-based logics [12]–[20]. Among them, Material Implication (IMPLY) [9], [21], [22] is one of the well-known logics which is compatible with crossbar structure [9], [21]. An important feature of IMPLY is that it allows presenting the logical values in the memory domain as well as performing logic operations inside memory within the crossbar array structure, thus colocating storage and processing. We note that IMPLY is not the only logic with such properties and there are other logics that have similar features. However, in the rest of this brief, we use IMPLY logic for our design and simulations.

In IMPLY [9], [21], [22], the resistance of the memristor represents the logical state, where R_{off} or high-resistance state (HRS) is considered as logic "0" and R_{on} or low-resistance state (LRS) as logic "1." In $a \rightarrow b$, IMPLY yields for all combinations of the two variables, a and b, the value of "1", except for $a =$ "1" and $b =$ "0". The result will be saved in b, i.e., b loses its initial value. IMPLY logic can be implemented with memristors as it is shown in Fig. 1. In Fig. 1, the inputs are the initial states of memristors a and b. These two are connected to a resistor, R_G. The output is the final state of memristor b after applying two fixed voltages, V_{SET} and V_{COND}, to memristors b and a, respectively. Basic conditions for building IMPLY gates are as follows [22]: $R_{on} << R_G << R_{off}$, $V_{\text{COND}} < V_C < V_{\text{SET}}$, and $V_{\text{SET}} - V_{\text{COND}} < V_C$, where V_C is the critical voltage (i.e., memristor threshold voltage), under which the memristor holds its initial state. This determines the voltages that should be applied to memristor. For further details on IMPLY operation, we refer the reader to [9], [21], and [22].

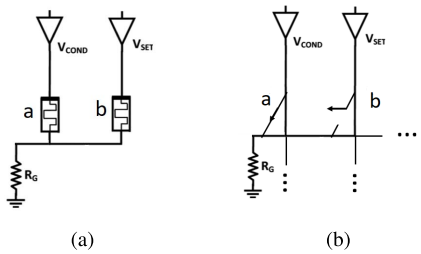

Fig. 1. (a) Circuit implementation of IMPLY logic gate. (b) Corresponding switch representation [9].

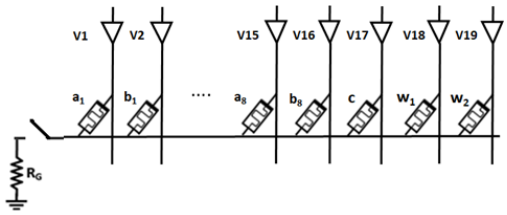

Fig. 2. Memristor connection in an 8-bit serial topology.

In the literature, two main approaches have been used for implementing IMPLY-based full-adders: serial (shown in Fig. 2) and parallel (shown in Fig. 3). In these figures, a and b represent the input, c the carry, and w_i the work memristors. In serial topology, V_m corresponds to the applied voltage for each individual memristor, whereas in parallel topology, where each section corresponds to a different bit, in $V_{n,m}$, n represents the bit index and m corresponds to the applied voltage for each individual memristor within the section for the nth bit.

The serial approach is the most common topology based on crossbar structure, in which all of the memristors including work, input and output stand in the same row, all connected to ground via a resistor (Fig. 2). In this approach, only one operation (IMPLY or FALSE) can be performed at a time. Consequently, this increases the total operation time. Many research works [21]–[25] use this structure.

The other approach, parallel topology [22], uses parallelism for reducing the operation time at the cost of increasing the total number of memristors. In this design, each bit stands in a different row with its related work memristors. The advantage of this model is that all independent operations can be executed simultaneously. However, there are many operations that are dependent, e.g., each bit needs to wait for the previous bit to provide its carry-out as the carry-in of the next bit. Moreover, each row works in serial, i.e., parallelism is possible only between independent operations of different bits, which do not require input from the previous bit. This restriction is one of the issues that we try to address differently in our semiparallel approach. Our contributions in this work can be summarized into a new topology (semiparallel), its respective new algorithm and equations, as well as six new equivalencies in IMPLY logic.

Manuscript received June 4, 2018; revised December 23, 2018, April 16, 2019, and July 1, 2019; accepted August 11, 2019. Date of publication September 11, 2019; date of current version December 27, 2019. *(Corresponding author: Nima Taherinejad.)*

S. G. Rohani was with TU Wien, 1040 Vienna, Austria. She is now with the Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854 USA (e-mail: shokouh.ganjei@gmail.com).

N. Taherinejad and D. Radakovits are with the Faculty of Electrical Engineering and Information Technology, Institute of Computer Technology, TU Wien, 1040 Vienna, Austria (e-mail: nima.taherinejad@tuwien.ac.at; david.radakovits@tuwien.ac.at).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2936873

1063-8210 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 18:40:05 UTC from IEEE Xplore. Restrictions apply.

TABLE I

EXECUTION STEPS OF ADDITION IN THE PROPOSED SEMIPARALLEL TOPOLOGY. THE STATE OF SWITCHES IN EACH STEP IS DENOTED BY A "0" FOR AN OPEN SWITCH AND "1" FOR A CLOSED (SHORT CIRCUIT) SWITCH

Steps	Operation executed in:			Switches (S1,S2,S3)	Equivalent Logic
	Section 1	Section 2	Between sections		
1	$w_1 = 0$	$w_2 = 0$		(1,0,1)	$FALSE(w_1) \quad FALSE(w_2)$
2	$a \to w_1 = w_1'$	$b \to w_2 = w_2'$		(1,0,1)	$\overline{a} \quad ; \quad \overline{b}$
3			$w_1' \to b = b'$	(1,1,0)	$\overline{a} \to b$
4			$a \to w_2' = w_2''$	(1,1,0)	$a \to \overline{b}$
5	a = 0			(1,0,0)	$FALSE(a)$
6			$b' \to a = a'$	(1,1,0)	$(\overline{\overline{a} \to b})$
7			$w_2'' \to a' = a''$	(1,1,0)	$\big((a \to \overline{b}) \to \overline{(\overline{a} \to b)}\big) = (\overline{a \oplus b})$
8	$w_1 = 0$			(1,0,0)	$FALSE(w_1)$
9			$c \to w_1 = w_1'$	(1,1,0)	$\overline{c}$
10	$a'' \to w_1' = w_1''$	$w_2'' \to c = c'$		(1,0,1)	$\big((a \to \overline{b}) \to \overline{(\overline{a} \to b)}\big) \to \overline{c} = (\overline{a \oplus b}) \to \overline{c} \quad ; \quad (a \to \overline{b}) \to c$
11	a=0	$w_2 = 0$		(1,0,1)	$FALSE(a) \quad FALSE(w_2)$
12	$w_1'' \to a = a'$	$c' \to w_2 = w_2'$		(1,0,1)	$\overline{\big((a \to \overline{b}) \to \overline{(\overline{a} \to b)}\big) \to \overline{c}} = \overline{(\overline{a \oplus b}) \to \overline{c}} \quad ; \quad \overline{(a \to \overline{b}) \to c}$
13		$b' \to w_2' = w_2''$		(0,0,1)	$(\overline{a} \to b) \to \overline{\big((a \to \overline{b}) \to c\big)}$
14		$b' \to c' = c''$		(0,0,1)	$(\overline{a} \to b) \to \big((a \to \overline{b}) \to c\big)$
15			$c'' \to a' = a''$	(0,1,1)	$\big[(\overline{a} \to b) \to \big((a \to \overline{b}) \to c\big)\big] \to \overline{(\overline{a \oplus b}) \to \overline{c}} = Sum$
16		c = 0		(0,0,1)	$FALSE(c)$
17			$w_2'' \to c = c'$	(0,1,1)	$\overline{(\overline{a} \to b) \to \overline{\big((a \to \overline{b}) \to c\big)}} = Carry - Out$

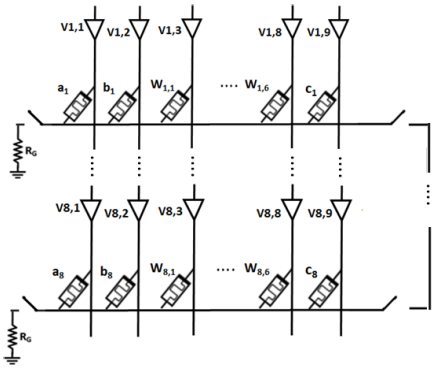

Fig. 3. Memristor connection in an 8-bit example using parallel topology. Each row calculates 1 bit.

II. PROPOSED FULL-ADDER DESIGN

Our main objectives for the new design are improvements in two aspects: reducing operation time and implementation area. However, the design process often comes down to a tradeoff between the two. For example, one way of reducing the delay time of a memristive IMPLY-based full-adder is increasing the number of memristors (enabling more parallelism). The latter is in conflict with the area reduction goal. Hence, given the disadvantages of the serial and parallel approaches, we were inspired to propose a new topology, which we call semiparallel. The idea of this design is to take advantage of the parallelization within 1 bit. This means that we divided operations needed for 1 bit of calculation (e.g., $a_0 + b_0$ or generally for the ith bit, $a_i + b_i$), into two independent sections, which can work parallel to each other as long as possible.

A. Sum and Carry-Out Logical Statements

In our algorithm, Sum (S) and Carry-out (C_{out}) are calculated using

$$S = [(\overline{a} \to b) \to ((a \to \overline{b}) \to c)] \to \overline{\overline{(\overline{a \oplus b}) \to \overline{c}}} \quad (1)$$

$$C_{\text{out}} = \overline{(\overline{a} \to b) \to \overline{(a \to \overline{b}) \to c}} \quad (2)$$

where a and b are the inputs and c is the Carry-in. To obtain them, we performed our heuristic logic minimization natively in IMPLY logic using the equivalencies in [25] and the following new equivalencies:

1. $a \to \overline{a} \equiv \overline{a}$ and $\overline{a} \to a \equiv a$
2. $(\overline{a \to c}) \to b \equiv \overline{c} \to (a \to b)$
3. $b \to (a \to \overline{c}) \equiv (\overline{b \to \overline{a}}) \to \overline{c}$
   (Similar to De Morgan's law in Boolean logic)
4. $(\overline{a} \to b) \to (a \to b) \equiv a \to b$
5. $(a \to b) \to (\overline{a} \to b) \equiv \overline{a} \to b$
6. $\overline{(\overline{a} \to \overline{b}) \to (\overline{C}ab)} \equiv (a \to \overline{b}) \to (\overline{\overline{a} \to b})$.

B. Semiparallel Topology

The proposed topology is illustrated in Fig. 4 for an 8-bit addition, where w_1 and w_2 are the work memristors and c is the carry bit. First, all input memristors will be set to their initial values, then the first bit is calculated. Within each bit, independent steps (such as FALSE operation of the two work memristors, w_1 and w_2) can be parallelized. In this model, each section can work autonomously as long as possible. Then, they can be connected through a switch (S_2) to perform operations which involve memristors from both sections. The efficiency of this topology increases when the executed algorithm for 1 bit has less dependence between different steps. The exact execution steps of the algorithm are detailed in Table I. The proposed algorithm requires 17 steps for calculating Sum and Carry-out. We note that in this structure, not every bit needs to be fully parallelized. That is, as shown in Fig. 4, all a memristors can be on the same line (Section I) in series, and all b memristors and c on the same line (Section II) in series. That makes the proposed structure crossbar compatible, where each section constitutes a column (or a row), and for each step, respective columns (or rows) are connected to the R_G or each other via external CMOS switches (S1, S2, and S3). The three drivers of each section (V_{SET}, V_{COND}, and V_{RESET}) are

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 18:40:05 UTC from IEEE Xplore. Restrictions apply.

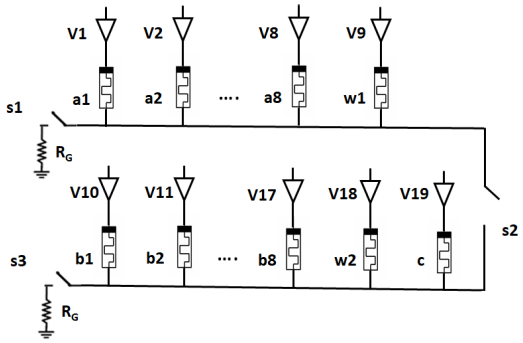

Fig. 4. Multi-bit semiparallel. Parallelization within 1 bit and calculating each bit after the other.

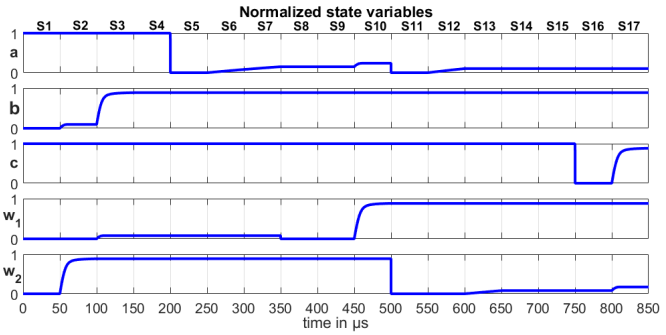

Fig. 5. 1-bit full addition using the semiparallel algorithm, where each 50μs represents one step in the algorithm. In this example, input $a = 1$, $b = 0$, $c = 1$ results in $c = 1$, $S = a = 0$, where 1 is LRS and 0 is HRS.

TABLE II
PARAMETER VALUES USED IN VTEAM

Parameter	v_{off}	v_{on}	α_{off}	α_{on}	R_{off}	R_{on}
Value	0.7 V	-10 mV	3	3	1 MΩ	10 kΩ
k_{on}	k_{off}	w_{off}	w_{on}	w_C	a_{off}	a_{on}
-0.5 nm/s	1 cm/s	3 nm	0 nm	107 pm	3 nm	0 nm

externally multiplexed between memristors of that section to apply appropriate voltages to the intended memristor.

C. Evaluation

1) Simulations Setup: To validate the proposed algorithm, we simulated it in LTSpice using the Voltage ThrEshold Adaptive Memristor (VTEAM) model [26]. The parameters used for the VTEAM are given in Table II and the SPICE implementation of the VTEAM can be found in [27]. Parameters for the IMPLY logic were set to $\{V_{\text{SET}}, V_{\text{COND}}, V_{\text{RESET}}, R_G, t_{\text{pulse}}\} = \{1\text{V}, 900\text{ mV}, -5\text{ V}, 40\text{ k}\Omega, 50\ \mu\text{s}\}$. In each step, V_{COND} is applied to the *antecedent* memristor and V_{SET} to the *consequent* memristor. No voltage is applied to memristors which are not involved in the operation of the respective step, i.e., they are disconnected from the source.

2) Results: We tested all the cases for a full-adder, all of which led to correct calculations. For example, in Fig. 5 we inserted the simulation of a 1-bit addition using the proposed semiparallel algorithm which shows the changes at each step. Input values in this example are $a = 1$, $b = 0$, Carry-in $= c = 1$, which results in Sum $= a = 0$ and Carry-out $= c = 1$. Fig. 6 shows the simulation of a 4-bit addition. Using the built-in power integration function of LTSpice, the energy consumption of the proposed adder was calculated to be 9.98 nJ per bit.

We note that as shown in Figs. 5 and 6, state variables sometimes undergo drift or do not reach 100% of their state (be it "0" or "1"). These two correlated cases occur mainly due to the fact that memristors experience small state changes even if the voltage applied to them is not larger than their threshold voltage. Moreover, crossing the threshold by itself does not guarantee a full state change. The difference between the applied voltage and the threshold as well as the duration of the applied voltage affect the extent of a state change. This phenomenon is observed, tested using four models, and discussed in [28] as well.

3) Crossbar Simulation: Although 1R crossbars are the most desired structures, currently they face many challenges which could be alleviated by using 1T1R crossbar architecture [29]–[31]. Given the advantages of 1T1R [29]–[31], this is the structure that we also chose for the proposed adder topology. Therefore, we simulated the addition of two 4-bit numbers with our proposed adder design in

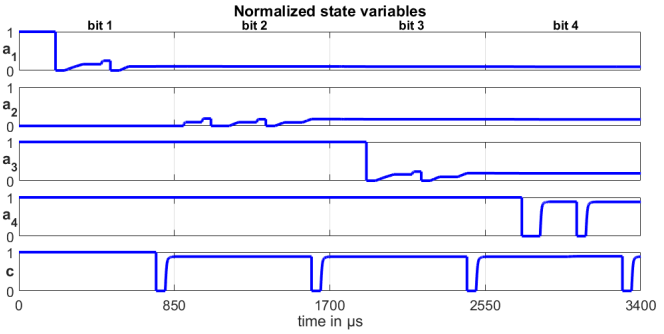

Fig. 6. 4-bit addition using the semiparallel algorithm, where each 850 μs represents calculation of 1 bit. In this example, input $a_{4-1} = 1101$, $b_{4-1} = 1010$, Carry-in $= c = 1$ results in Carry-out $= c = 1$, $s = a_{4-1} = 1000$.

a 16×16 1T1R crossbar. In this particular simulation, based on the work in [13], [33], and [34], we considered the crossbar connections between individual memristor nodes to have an ohmic impedance of 10Ω. This ohmic impedance represents the resistance of the nanowires in the crossbar. The results of this simulation (i.e., the state of the involved memristors after the calculation is completed) have shown a difference of only 0.52% on average and less than 2% in all cases compared to the stand-alone simulation. In other words, there is only a negligible difference between crossbar simulation and the one using only the 11 necessary memristors for the addition of two 4-bit numbers.

4) Sensitivity Analysis: In light of unideal behaviors such as those mentioned above, we ran a sensitivity analysis on our adder regarding changes in the threshold voltage of the memristor. Our simulation results show that the proposed adder functions correctly for positive and negative threshold ranges of $[595, 775]$ mV and $[-95, 0]$ mV, respectively.

5) Limitations: Our current simulation is at the behavioral level. Certain practical considerations such as parasitic models are not fully considered since we have not targeted a specific fabrication technology nor have access to a more comprehensive realistic model which includes those factors. We acknowledge that we need to consider such issues and many more [28] in order to ensure the full practicality of the design. However, due to the lack of access to better simulation tools or fabrication, following many other state-of-the-art works which are validated similarly, these aspects are left out and are considered as future works.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 18:40:05 UTC from IEEE Xplore. Restrictions apply.

TABLE III
SUMMARY OF COMPARISONS BETWEEN THE PROPOSED ALGORITHM AND OTHER IMPLY-BASED WORKS FOR $n = 32$ (A 32-BIT FULL-ADDER)

Designs	Number of Memristors							Number of Steps				
	Input	Output	Reused	Work	Total	$n = 32$	Imp.$^+$	IMPLY	FALSE	Total	$n = 32$	Imp.$^+$
Serial [24]	$2n+1$	$n+1$	1	2	$3n+3$	99	33%	$15n$	$8n$	$23n$	736	26%
Serial [22]	$2n+1$	$n+1$	1	2	$3n+3$	99	33%	$19n$	$10n$	$29n$	928	41%
Serial [25]	$2n+1$	$n+1$	$n+1$	2	$2n+3$	67	0%	$15n$	$7n$	$22n$	704	23%
Parallel [22]	$2n+1$	$n+1$	1	$6n$	$9n$	288	78%			$5n+18$	178	-70%
Iterative [33]	$3n$	n	n	$5n$	$8n$	256	75%			$21n-3$	669	19%
Proposed	$2n+1$	$n+1$	$n+1$	2	$2n+3$	67	-	$12n^*$	$5n^*$	$17n$	544	-

$^+$ Percentage of improvement (Imp.) is calculated based on $n \to \infty$.
* The number of steps for IMPLY and FALSE indicate the number of each operation, needing a separate cycle (to be representative of the total number of steps). That is, operations that are meant to be run in parallel are counted only as one step.

III. COMPARISON

To have a better understanding of the advantages and disadvantages of the proposed algorithm, we compared our design with the best existing works in the literature. In Table III, we have summarized the design characteristics for the proposed design and the competing designs. Percentages of improvement are calculated based on $(P_{\text{base}} - P_{\text{better}}/P_{\text{base}}) \times 100$, where P_{better} is the better design and P_{base} is the base for comparison.

A. Speed

The proposed semiparallel design needs only 544 steps to finish a 32-bit addition, which is 19% better than the iterative design [33], 23% better than our previously published serial design [25], 26% faster than [24], and 41% better than Kvatinsky's serial design [22]. However, the 544 steps are 67% slower than fully parallel designs of Kvatinsky [22]. For $n \to \infty$, this number would approach 70%. This loss of speed comes with the advantage of a considerable (78%) improvement in the required number of memristors, which we will discuss in the following.

B. Area

Chip area directly translates to the production cost and, therefore, plays an important role in determining the merit of a design and its potential for widespread use and implementation. Hence, with extra attention to this factor, we have managed to propose the most compact design thus far. For example, in the case of a 32-bit full-adder, our design—similar to our previously published serial design [25]—only needs 67 memristors ($2n+3$), whereas other serial designs [22], [24] need 99 memristors ($3n + 3$). By increasing the number of bits, this improvement approaches 33%. Regarding the number of memristors, the iterative design in [33], even though based on its name it may sound as if it were a serial design, is more similar to parallel designs, since it needs 256 memristors. This design requires 75% more memristors than our proposed design. Compared to parallel designs, 67 memristors needed in our design is significantly (78%) lower than the 288 memristors ($9n$) needed in Kvatinsky's fully parallel design [22], which justifies the smaller (70%) degradation in the number of steps.

C. Complexity

The additional complexity of this structure compared to a serial structure is only the three CMOS switches which connect the two sections to the IMPLY load resistors and to each other. This is a minimal overhead for the traditional serial structure. Compared to the parallel topology, which requires two switches for each row of memristors, the proposed structure requires a significantly smaller number of switches. In particular, consider that the number of switches in the parallel structure increases in proportion to the width of the adder ($2n$ in an n-bit adder), whereas in the proposed structure this number is a constant (only three switches for any n-bit adder). Furthermore, the proposed adder algorithm runs in serial steps one bit after the other. Hence, the state machine and the control necessary to run this algorithm are similar to the serial topology, with three additional outputs for controlling the switches in each state. On the other hand, a smaller number of steps denotes a smaller number of states and a state machine simpler than the traditional serial topology. We expect the required control circuit to be much less complex than the parallel topology.

D. Limitations

We notice that our comparisons have certain restrictions. For example, the actual computation time and energy consumption heavily depend on the type of used memristors and in our case the used model and its parameters. With regard to the actual computation time, it is reasonable to assume that in the exact same technology, an algorithm with a smaller number of steps is going to require a shorter absolute calculation time as well. Although the number of memristors is a good indicator for the required area, additional factors such as the area used by the control circuit and switches need to be considered for a comprehensive comparison. The details of the memristive technology and the peripheral circuits are even more important for a fair comparison of energy consumption. To achieve a comprehensive and fully fair comparison, fabrication or a postlayout simulation—in the same technology—is required. None of the references have fabricated or reported postlayout simulation results to form the base of such a comparison. This is not possible for us either.

IV. CONCLUSION

In this brief, we proposed a new full-adder structure that differs from both serial and parallel topologies, which are common in the literature. We call this topology semiparallel. Our adder managed to surpass serial designs by up to 78% in area and up to 41% in speed. Although our design is 70% slower than its parallel counterpart, it achieves 78% of smaller area advantage. Therefore, in a generic condition, where area and speed are of equal importance, we contend that this design is better than the existing fully parallel design. Otherwise, the design situation at hand needs to be considered to evaluate whether the proposed algorithm is more suitable or not. However, it is fair to say that this algorithm provides a good compromise between the speed and area (compared to each of the serial or parallel designs).

It is worth noting that till date IMPLY-based adders are designed heuristically. Even though there is no formal guarantee regarding

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 18:40:05 UTC from IEEE Xplore. Restrictions apply.

the optimum number of steps or memristors, the state of the art seems very close to a saturated state which could be (close to) its optimum. This observation is backed by the trend of improvements, e.g., in serial adders. The algorithm proposed in [21] required a total of 89 steps and four work memristors for a single bit addition. After a relatively big jump in 2014 to $29n$ steps and three work memristors for an n-bit addition [22], next improvements have been minor. This presents the challenging nature of the task at hand. The number of steps was reduced to $23n$ without changing the number of memristors in 2014 [24] and to $22n$ steps with the reduction of memristors from $3n+3$ to $2n+3$ in 2017 [25]. In this relatively optimum and saturated state, every reduction is of considerable value. Hence, we contend that our proposed algorithm which requires $17n$ steps and $2n + 3$ memristors for an addition has a noteworthy contribution.

REFERENCES

[1] Y. Ho, G. M. Huang, and P. Li, "Dynamical properties and design analysis for nonvolatile memristor memories," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 4, pp. 724–736, Apr. 2011.

[2] V. Baghel and S. Akashe, "Low power memristor based 7T SRAM using MTCMOS technique," in *Proc. ACCT*, Feb. 2015, pp. 222–226.

[3] M. Zangeneh and A. Joshi, "Design and optimization of nonvolatile multibit 1T1R resistive RAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 8, pp. 1815–1828, Aug. 2014.

[4] N. Taherinejad, P. D. S. Manoj, and A. Jantsch, "Memristors' potential for multi-bit storage and pattern learning," in *Proc. EMS*, Oct. 2015, pp. 450–455.

[5] N. Taherinejad, P. D. S. Manoj, and A. Jantsch, "Fully digital write-in scheme for multi-bit memristive storage," in *Proc. CCE*, Sep. 2016, pp. 1–6.

[6] Y. V. Pershin, S. L. Fontaine, and M. D. Ventra, "Memristive model of amoeba learning," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 80, Aug. 2009, Art. no. 021926.

[7] A. Thomas, "Memristor-based neural networks," *J. Phys. D, Appl. Phys.*, vol. 46, no. 9, Feb. 2013, Art. no. 093001.

[8] Y. V. Pershin and M. Di Ventra, "Neuromorphic, digital, and quantum computation with memory circuit elements," *Proc. IEEE*, vol. 100, no. 6, pp. 2071–2080, Jun. 2012.

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

[10] V. Hongal, R. Kotikalapudi, and M. Choi, "Design, test, and repair of MLUT (memristor look-up table) based asynchronous nanowire reconfigurable crossbar architecture," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 4, no. 4, pp. 427–437, Dec. 2014.

[11] A. Zaher, P. Häfliger, F. Puppo, G. De Micheli, and S. Carrara, "Novel readout circuit for memristive biosensors in cancer detection," in *Proc. BioCAS*, Oct. 2014, pp. 448–451.

[12] L. Gao, F. Alibart, and D. B. Strukov, "Programmable CMOS/memristor threshold logic," *IEEE Trans. Nanotechnol.*, vol. 12, no. 2, pp. 115–119, Mar. 2013.

[13] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Mater.*, vol. 9, no. 5, pp. 403–406, Apr. 2010.

[14] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Beyond von Neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, Jul. 2012, Art. no. 305205.

[15] S. Kvatinsky *et al.*, "MAGIC—Memristor-aided logic," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.

[16] I. Vourkas and G. Sirakoulis, "Memristor-based nanoelectronic computing circuits and architectures," in *Emergence, Complexity and Computation*. Cham, Switzerland: Springer, 2015. doi: 10.1007/978-3-319-22647-7.

[17] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 5, no. 1, pp. 64–74, Mar. 2015.

[18] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, Jul. 2016.

[19] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *Proc. ISVLSI*, Jul. 2017, pp. 176–181.

[20] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Solids Surf.*, vol. 80, no. 6, pp. 1165–1172, Mar. 2005.

[21] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. NANOARCH*, Jul. 2009, pp. 33–36.

[22] 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.

[23] K. A. Bickerstaff and E. E. Swartzlander, "Memristor-based arithmetic," in *Proc. ACSSC*, Nov. 2010, pp. 1173–1177.

[24] M. Teimoory, A. Amirsoleimani, J. Shamsi, A. Ahmadi, S. Alirezaee, and M. Ahmadi, "Optimized implementation of memristor-based full adder by material implication logic," in *Proc. ICECS*, Dec. 2014, pp. 562–565.

[25] S. G. Rohani and N. Taherinejad, "An improved algorithm for IMPLY logic based memristive full-adder," in *Proc. CCECE*, Apr./May 2017, pp. 1–4.

[26] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 62, no. 8, pp. 786–790, Aug. 2015.

[27] (Mar. 2018). *Spice Implementation of VTEAM Model*. [Online]. Available: https://www.ict.tuwien.ac.at/staff/taherinejad/projects/memristor/files/VTEAM.sub

[28] N. Taherinejad and D. Radakovits, "From behavioral design of memristive circuits and systems to physical implementations," *IEEE Circuits Syst. Mag.*, to be published.

[29] H. J. Wan *et al.*, "*In situ* observation of compliance-current overshoot and its effect on resistive switching," *IEEE Electron Device Lett.*, vol. 31, no. 3, pp. 246–248, Mar. 2010.

[30] C. Li *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electron.*, vol. 1, no. 1, pp. 52–59, Dec. 2018.

[31] C. Li *et al.*, "In-memory computing with memristor arrays," in *Proc. IEEE IMW*. May 2018, pp. 1–4.

[32] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. H. Fahmy, and K. N. Salama, "Memristor multiport readout: A closed-form solution for sneak paths," *IEEE Trans. Nanotechnol.*, vol. 13, no. 2, pp. 274–282, Mar. 2014.

[33] K. C. Rahman, M. R. Khan, and M. A. Perkowski, "Memristor based 8-bit iterative full adder with space-time notation and sneak-path protection," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 695–698.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 18:40:05 UTC from IEEE Xplore. Restrictions apply.