IMPLY-Based High-Speed Conditional Carry and Carry Select Adders for In-Memory Computing

Nandit Kaushik, *Graduate Student Member, IEEE*, and Srinivasu Bodapati, *Member, IEEE*

***Abstract*—Big data applications involved with neural networks requires frequent data transfer between memory and processing elements and thus, take a significant portion of energy. One possible method is to realize logic operations inside the memory, often known as *in-memory computing*. The memristor employed in a crossbar can also conduct logic operations, making it a promising candidate for enabling such designs. In-memory computing can vastly improve speed efficiency by significantly reducing memory access time and energy. Therefore, efficient operations inside the memory are crucial for data-intensive applications. This article presents two high-speed adders in memristive technology, which are derived using IMPLY-based logic operations and can be implemented on a crossbar architecture for in-memory computing. One is a carry select adder, which uses efficient ripple carry adders and multiplexers. While the other is a conditional carry adder, designed using a modified half adder and multiplexers. An IMPLY-based multiplexer and modified half adder are proposed as part of these adders. A modified half adder calculates the sum of two inputs and two carry outputs for various carry inputs. Compared to conventional adder designs, the speed of these two adders has been significantly enhanced. In particular, the proposed 32-bit Carry Select Adder and Conditional Carry Adder take 123 and 90 computational steps, respectively. The proposed adders have improved the speed by 30% and 49%, respectively, compared to the best-existing adder in IMPLY. Further, this article presents computational steps and the number of memristors required for n-bit adders. The proposed circuits were simulated using the VTEAM model, and energy consumed for each adder is calculated from the simulations.**

***Index Terms*—Adder, carry select, conditional carry, memristor, imply and emerging technologies, in-memory computing.**

I. Introduction

With the emergence of IoT applications, algorithms such as image detection, image processing, and speech recognition in intelligent applications have experienced an increase in data, necessitating a greater number of computations to process. A deep neural network can be utilized for the efficient implementation of the circuits for data-intensive applications. The continuous memory access between the data path and memory leads to a major portion of time and energy consumption [1]. This is a major concern in computations that involve big-data driven applications. The energy and time consumed for frequent memory access are dominant in a conventional von-Neumann system [2], popularly known as “Von-Neumann Bottleneck” [3]. The most typical approach is near-memory computing by incorporating several cache memory levels next to the CPU [4]. Cache memories drastically reduce the amount of data transfer between the processing unit and memory [5]. There exist various FPGA-based near-memory architectures for neural networks, where the major portion of the energy is consumed in the memory access between the processing element and the on-board SRAM [6]. Several works have investigated the idea of merging the processing unit within the memory [7], [8]. However, it still requires data transfers between processing and memory units and thus, cannot fully utilize the advantages of in-memory computing. Moreover, regular memory cells are not well adapted for direct execution of computations between memory cells.

Emerging device technologies, including Memristor [9] and spintronic-based Magnetic Random Access Memory (MRAM) [10], are being explored as potential solutions to overcome the existing device limitations by enabling data storage and logical operations within the same device. Such devices can be used for in-memory and near-memory computing. Memristors use their resistance to store binary data. *High* and *low* resistance represents logic ‘0’ and logic ‘1’ respectively. The resistance can switch between high and low on applying suitable voltages and provide significant advantages over existing cell technologies due to their high density, non-volatility, low power consumption, and compatibility with CMOS production [11]. Using stateful logic [12], [13], in which the logic states of logic gates are represented by resistance, is an appealing technique of executing logic in a memristive array. Memristive devices can be used as a switch, programmable interconnect and computational device for implementing neural network circuits [14], [15]. There exist various logic methodologies to design memristive-based arithmetic and logical circuits such as Memristive-Resistive Logic (MRL) [16], CMOS-Memristor logic [17], Memristor-Aided-Logic (MAGIC) [18], and Material Implication (IMPLY) logic [19]. Due to its comparatively simple logic structure and ease of implementation in crossbar architecture, IMPLY-based logic design is commonly employed for high-density device designs.

Among the fundamental components of every processor, adders are vital since they are employed in every single instruction. Further, in neural networks and neuromorphic computing circuits, multiply accumulation units require a huge number of addition operations. Therefore, efficient adders are required in memristive technology with in-memory computing capability.

Manuscript received 21 April 2023; accepted 5 June 2023. Date of publication 12 June 2023; date of current version 22 June 2023. The review of this article was arranged by Associate Editor J. S. Friedman. *(Corresponding author: Nandit Kaushik; Srinivasu Bodapati.)*

The authors are with the School of Computing and Electrical Engineering, Indian Institute of Technology, Mandi 175005, India (e-mail: knandit312@gmail.com; srinivasu@iitmandi.ac.in).

Digital Object Identifier 10.1109/TNANO.2023.3284845

1536-125X © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

TABLE I
TRUTH TABLE FOR X → Y

x	y	$x \to y$
0	0	1
0	1	1
1	0	0
1	1	1

Various Ripple Carry Adders (RCAs) in IMPLY logic are presented in literature [19], [20], [21], [22], [23], [24], [25] in serial and parallel approaches. Adders in serial topology [19], [20], [21] have better compatibility in a crossbar implementation, which requires fewer memristors and more computational steps. As an alternative, parallel topology [19], [23], [24], [25] reduces the computational steps by performing independent operations simultaneously. In this approach, input memristors are placed on independent rows along with the work memristors for parallel computations, which leads to increased number of memristors. The parallel computations in a regular 2D memristive crossbar requires insertion of switches for selecting memristors from a particular row and column for simultaneous IMPLY operations. Hence these parallel architectures are implemented in a 1T1R crossbar array [24].

This article presents two alternative adder designs, the IMPLY logic-based Carry Select Adder (CSA) and Conditional Carry Adder (CCA), which do faster additions compared to the traditional RCA. The contributions of the proposed designs are as follows. The proposed CSA employs an efficient parallel architecture consisting of an efficient RCA and multiplexers (MUXs). An IMPLY-based 2×1 multiplexer is introduced, requiring five memristors and five computational steps. For a 32-bit addition, the proposed CSA takes 123 computational steps compared to the 176 required with the best existing RCA design [25]. A Modified Half Adder (MHA) is proposed as a part of the proposed CCA, which computes one *sum* output S^* and two *carry* outputs C^0 and C^1 (based on C_{in} = '0' and C_{in} = '1'), in 11 computational steps. Using these MHAs and MUXs, a $32 - bit$ addition is performed through the proposed CCA in 90 computational steps. Compared to existing RCA designs, the proposed CSA enhances the addition speed by 83.2% [20] to 30.1% [25], whereas the proposed CCA sees an improvement in the range of 87.7% [20] to 48.8% [25]. Further, this article presents the expressions for computational steps and number of memristors for proposed n-bit adders.

The rest of the article is organized as follows. Section II presents IMPLY logic and existing adders in IMPLY logic. The proposed Carry Select Adder and Conditional Carry Adder are presented in Sections III and IV respectively. The simulation results are presented in Section V. Section VI gives the comparisons and Section VII concludes the article.

II. IMPLY Logic and Adders in IMPLY Logic

The IMPLY logic designed using memristors is shown in Fig. 1. The IMPLY operation between x and y is denoted as $x \to y$ (read as $x\ imply\ y$) and is defined via truth table in Table I. The IMPLY operation $x \to y$ can be simplified as $\overline{x} + y$.

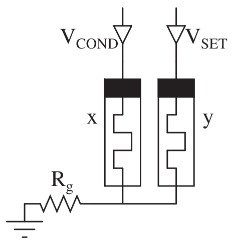

Fig. 1. IMPLY logic using memristors.

The initial states of the memristors x and y act as the input. The final memristance of y is the output, after the simultaneous application of voltages V_{COND} and V_{SET}.

There exist various memristor-based one-bit adders in IMPLY logic in serial and parallel approaches. A serial architecture based n-bit RCA is implemented with $3n + 5$ memristors and takes $89n$ computational steps [12]. Authors in [19] reported an n-bit RCA by reusing the input memristors and computing the C_{out} from the intermediate result of *SUM* in $29n$ computational steps with $3n + 3$ memristors. A parallel implementation of n-bit RCA proposed in [19] takes fewer computational steps as $5n + 18$ with $9n$ memristors. Furthermore, there are studies that aim to resolve the tradeoff between the number of memristors and the required number of computations. Authors in [21] proposed an efficient serial implementation of n-bit RCA using $22n$ computing steps and $2n + 3$ memristors. IMPLY-based optimized n-bit adder with parallel computations requires $4n + 1$ memristors and takes $5n + 16$ computational steps [25]. Further, a combination of the serial and parallel approach known as the semi-serial approach of n-bit RCA requires $10n + 2$ computational steps and $2n + 6$ memristors [23]. A semi-parallel adder in [24] takes $2n + 3$ memristors to compute the result in $17n$ computing steps.

All of the adders reported in the literature were ripple carry-based architectures. In this article, we propose two fast adders as *Carry Select Adder* and *Conditional Carry Adder* in IMPLY logic, which does the parallel computations of the intermediate *sum* and *carry* in advance and then select the corresponding results using multiplexers. In particular, CSA requires ripple carry adders and multiplexers while CCA requires one-bit modified half adders and multiplexers. In next section, we propose a *Carry Select Adder* using IMPLY logic.

III. Carry Select Adder

Fig. 2 depicts a 8-bit *Carry Select Adder* using 4-bit RCA and multiplexers [26]. In particular, the 8-bit CSA requires three 4-bit RCAs and five 2×1 MUXs. The first RCA module generates the *SUM* and *Carry Out* for input C_{in}. While the other two RCA modules generate the *SUM* and the leading carry considering C_{in} = '0' and '1', respectively. The final output is selected using multiplexers as shown in Fig. 2. The proposed CSA uses the ripple carry adder design using the full adder proposed in [25], which takes $5n + 16$ computational steps for n-bit addition. Further, we propose a 2×1 multiplexer in IMPLY logic, which will be used in designing efficient carry select adder.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

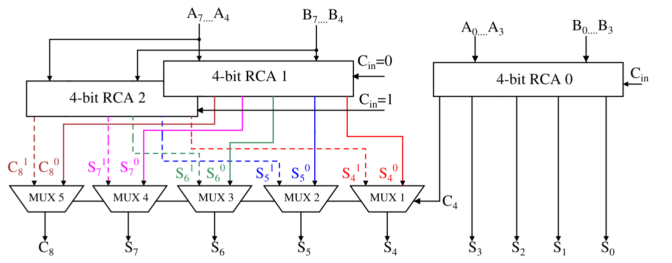

Fig. 2. 8-bit Carry Select Adder; A, B are two 8-bit inputs, C_{in} is the input carry; S is the 8-bit SUM while C_8 is the final carry output.

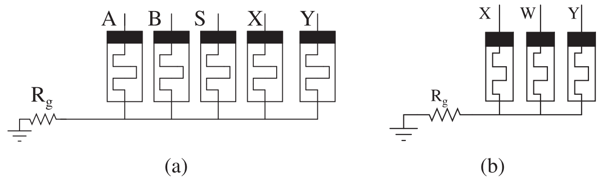

Fig. 3. IMPLY-based memristor circuits: (a) 2×1 MUX. (b) Copy operations.

TABLE II
COMPUTATIONAL STEPS FOR PROPOSED 2×1 MUX IN IMPLY LOGIC

Steps	IMPLY Operation	Equivalent Logic
1	$S \rightarrow X$	$X = \overline{S}$
2	$B \rightarrow X$	$X = \overline{B} + \overline{S}$
3	$A \rightarrow S$	$S = \overline{A} + S$
4	$S \rightarrow Y$	$Y = A\overline{S}$
5	$X \rightarrow Y$	$Y = \overline{X} + Y = BS + A\overline{S}$

A. IMPLY Based 2 × 1 Multiplexer

A 2×1 multiplexer implemented in IMPLY logic is depicted in Fig. 3(a), which is designed using five memristors, two input memristors (A, B), one select line memristor (S) and two work memristors (X, Y). The $2 \times 1\, MUX$ output is stored in the memristor Y. The proposed multiplexer takes five computational steps as shown in Table II.

B. 8-Bit Carry Select Adder in IMPLY Logic

Fig. 4 presents a memristive architecture for the proposed 8-bit CSA. The two 8-bit inputs are named as $A_0, A_1, \ldots, A_7$ and $B_0, B_1, \ldots, B_7$. The operation of the carry select adder can be categorized into three phases. Firstly, the generation of intermediate *sum* and *carry* using ripple carry adders. Next, performing copy steps for further computations. Lastly, selecting the final *SUM* and $Carry\, Out$ using multiplexers.

Sum and Carry Generation using RCA

All the work memristors are initialized to ‘0’. The proposed architecture is able to perform parallel computations to generate the output *sum* and *carry* of the 4-bit RCA. Each 4-bit RCA computes the *sum* as $(S_{0-3}, S^0_{4-7}$ and $S^1_{4-7})$ and C_{out} as $(C_4, C^0_8$ and $C^1_8)$ in 36 computational steps. The sum S_0 of A_0, B_0 is stored in the B_0 memristor. Similarly, the sum outputs $S^0_4, S^0_5, \ldots, S^0_7, S^1_4, S^1_5, \cdots S^1_7$ are stored in the memristor B in their respective rows. The three carry outputs C_4, C^0_8 and C^1_8 from the three 4-bit RCA are stored in the corresponding C memristors as depicted in Fig. 4.

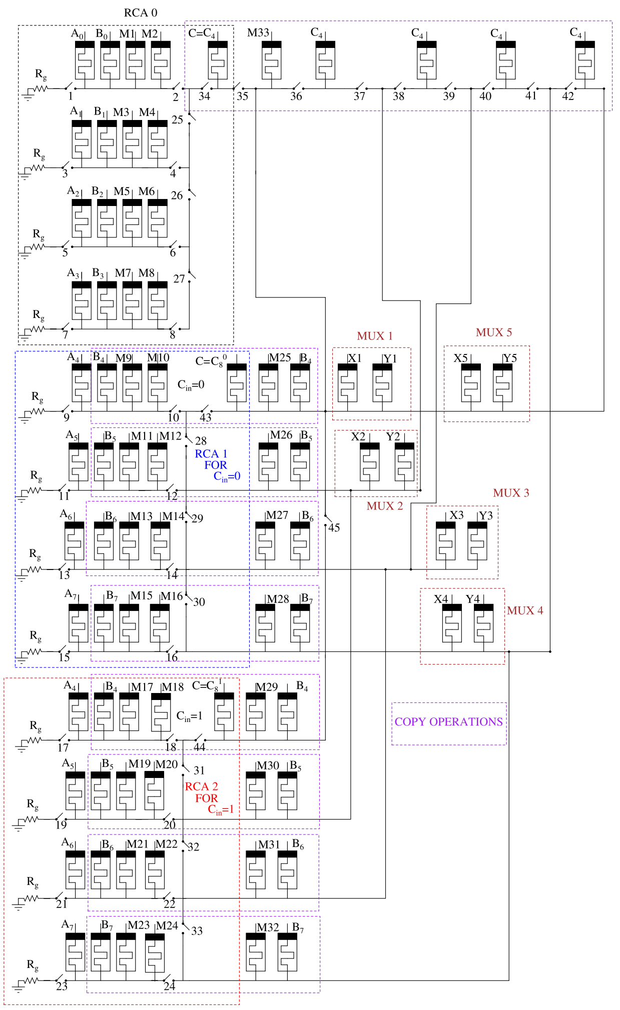

Fig. 4. Proposed 8-bit Carry Select Adder using memristors in IMPLY logic.

Copy Operations

Once the *sum* and *carry* outputs are computed, the carry output (C_4) from *RCA-0* will be used as the select signal to select sum and carry outputs of the MSB 4-bits. Five parallel multiplexers are required to select sum (S_{4-7}) and carry (C_8) using a single select line (C_4). Hence, the carry C_4 needs to be copied four times, while $S^0_{7:4}$ and $S^1_{7:4}$ are copied once for multiplexer operations. All these copy operations are performed simultaneously in five computational steps.

The circuit shown in Fig. 3(b) with three memristors is used to copy a value from memristor X to V through IMPLY operations. A copy operation takes two computational steps as shown in Table III.

Final Sum and Carry Selection using Multiplexer

Table IV explains the selection of final sum (S_{4-7}) and carry (C_8) through multiplexers. Five multiplexers are required to

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

TABLE III
COPY STEPS IN IMPLY LOGIC

Steps	IMPLY Operation	Equivalent Logic
1	$X \rightarrow W$	$W = \overline{X}$
2	$W \rightarrow V$	$V = X$

TABLE IV
MUX OPERATIONS FOR SELECTING SUM S_{4-7} AND C_8

ON Switches	MUX Operations			Memristors Used
	MUX	I/P	O/P	
9, 10, 35, 43, 45	1	S_4^0, S_4^1, C_4	S_4	B_4, B_4, C_4, X1, Y1
37, 11, 12	2	S_5^0, S_5^1, C_4	S_5	B_5, B_5, C_4, X2,Y2
39, 13, 14	3	S_6^0, S_6^1, C_4	S_6	B_6, B_6, C_4, X3,Y3
41, 15, 16	4	S_7^0, S_7^1, C_4	S_7	B_7, B_7, C_4, X4,Y4
9, 10, 43, 45	5	C_8^0, C_8^1, C_4	C_8	C_8^0, C_8^1,C_4, X5,Y5

TABLE V
SUMMARY OF COMPUTATIONAL STEPS FOR 8-BIT CSA

S.No	Operation	No. of Steps	No. of Memristors
1	4-bit RCA addition	1-36	17*3= 51
2	Copying of C_4, $S^0_{4\ldots7}$ & $S^1_{4\ldots7}$	37-41	21
3	Selection of sum S_7, S_6, S_5, S_4 through four 2×1 MUX	42-46	8
	Selection of C_8 using 2×1 MUX	47-51	2
Total		51	82

select the four sum and one carry output. Table IV lists the switches used to switch ON a multiplexer. Considering the switches 9, 10, 35, 43 and 45 are turned *ON*, it results the multiplexer MUX-1 to switch ON and selects the S_4 from S_4^0, S_4^1 using select line C_4. The multiplexers (MUX-1, MUX-2, MUX-3 and MUX-4) are placed in different rows to compute sum (S_{4-7}) parallelly (as shown in Fig. 4) in five computational steps. Similarly, the carry(C_8) can be selected through MUX-5 as shown in Table IV. Altogether, proposed 8-bit CSA MUXs selection requires 10 computational steps.

The performance metrics of the proposed 8-bit CSA are summarized in Table V. The computational steps and the number of memristors are categorized into three as RCA addition, copying and final output selection. In particular, 8-bit CSA requires 51 computational steps and 82 number of memristors.

In the next section, we derive the architecture and performance metrics of n-bit carry select adder.

C. Proposed n-Bit Carry Select Adder in IMPLY Logic

The proposed n-bit CSA is depicted in Fig. 5, which requires three $\frac{n}{2}$-bit RCA and few multiplexers. The $RCA\,0$ calculates the sum and $carry$ of the least significant $\frac{n}{2}$-bits as $S_{0,\ldots,n/2}$ and $C_{n/2}$ respectively as shown in Fig. 5. Whereas, the $RCA\,1$ and $RCA\,2$ calculates the sum and $carry$ by considering C_{in} as '0' and '1' as illustrated in Fig. 5. A series of $2{\times}1$ MUXs are used to select the sum $S_{n/2,\ldots,n-1}$ and carry C_n using the carry output of the $RCA\,0$ ($C_{n/2}$) as select line of the multiplexer. The memristor-based architecture for 8-bit CSA shown in Fig. 4 can be expanded to n-bit CSA. Next, we will derive the number

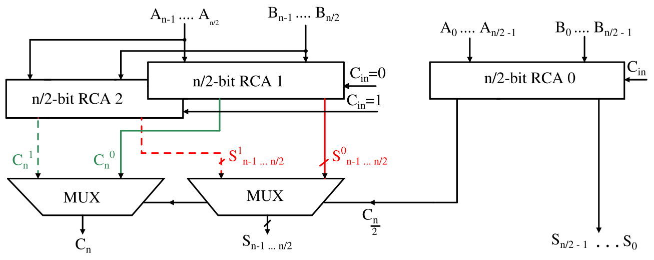

Fig. 5. n-bit Carry Select Adder using $\frac{n}{2}$ ripple carry adder.

of computational steps and memristors required to implement n-bit CSA using $\frac{n}{2}$ RCA through Propositions 1 and 2.

Proposition 1: The proposed IMPLY-based n-bit CSA requires $19\frac{n}{2} + 6$ memristors.

Proof: The proposed n-bit CSA is implemented using three $\frac{n}{2}$-bit RCA. Each $n/2$-bit RCA requires $4(n/2) + 1$ number of memristors. Hence, three RCA's take $3 \times (4\frac{n}{2} + 1)$ number of memristors. Next, the number of memristors required for copying the corresponding carry and sum can be calculated as follows. The carry output from *RCA 0* ($C_{n/2}$) need to be copied $\frac{n}{2} + 1$ times to select the most significant $\frac{n}{2}$ sum outputs and one carry (C_n) output. Hence, $\frac{n}{2} + 1$ memristors are required to copy $C_{n/2}$. The sum outputs from *RCA 1* and *RCA 2* need to be copied once, hence $S^0_{n/2,\ldots,n-1}$ and $S^1_{n/2,\ldots,n-1}$ in total requires $4 \times \frac{n}{2}$ memristors. Finally, the proposed n-bit CSA requires $(\frac{n}{2} + 1)$ multiplexers to select the $\frac{n}{2}$-sum and one carry output. The proposed multiplexer takes additional two memristors (input memristors are already present), as shown (marked in dashed box) in Fig. 4. Hence, multiplexers require $2 \times (\frac{n}{2} + 1)$ memristors. Altogether, the number of memristors required to implement n-bit CSA given in (1).

$$No.\,of\,Memristors = 3 \times \left(4\frac{n}{2} + 1\right) + \left(\frac{n}{2} + 1\right) + 4 \times \left(\frac{n}{2}\right) + 2 \times \left(\frac{n}{2} + 1\right) \quad (1)$$

$$No.\,of\,Memristors = 19\frac{n}{2} + 6 \quad (2)$$

As per (2), the proposed n-bit CSA takes '$19\frac{n}{2} + 6$' memristors. Next, we derive the number of computational steps required for IMPLY-based n-bit CSA through Proposition 2.

Proposition 2: An IMPLY-based n-bit CSA requires $3n + 27$ computational steps.

Proof: The number of computational steps required to compute n-bit CSA are as follows. First, the $\frac{n}{2}$-RCA computes the sum and carry in $(5\frac{n}{2} + 16)$ computational steps. Two independent copy operations were performed in parallel. Firstly, copying of $C_{n/2}$ for $\frac{n}{2}$ times, take $\frac{n}{2} + 1$ computational steps and one time copying of $\frac{n}{2}$-sum outputs. Hence, a total of $\frac{n}{2} + 1$ computational steps for copying. Finally, two multiplexer operations to select $S_{n/2,\ldots,n-1}$ and C_n, each multiplexer takes five steps. Altogether, the number of computational steps for

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

n-bit carry select adder can be calculated as given in (3).

$$No.\,of\ Computational\ Steps = 5\left(\frac{n}{2}\right) + 16$$
$$\frac{n}{2} + 1 + 10 \qquad (3)$$
$$No.\,of\ Computational\ Steps = 3n + 27 \qquad (4)$$

From (4), the proposed n-bit carry select adder requires '$3n + 27$' number of computational steps.

The advantages of the proposed CSA are given in Remark 1.

Remark 1: The proposed carry select adder takes less computational steps compared to the existing ripple carry adder with an increase in number of memristors because of the parallel computations. In particular, the proposed 32-bit CSA designed using 16-bit RCA takes 123 computational steps compared to the 176 steps needed by the best existing RCA design reported in [25].

In the next section, we propose *Conditional Carry Adder* in IMPLY logic.

IV. Conditional Carry Adder

The carry select adder uses an efficient ripple carry adder to design a high speed adder. The speed of the adder can be increased by reducing the size of RCA ($\frac{n}{2}$). Two popular adders which perform bit-wise additions by considering various C_{in} are *conditional sum adder* [27] and *conditional carry adder* (CCA) [28]. Conditional sum adder generates sum and carry for individual bits for various C_{in} while CCA generates carry outputs for various C_{in}. Next, we present a CCA design using memristors in IMPLY logic.

The working of conditional carry adder shown in Fig. 6 is as follows. The modified half adder (MHA) generates the sum and carry outputs in parallel for individual inputs. Then, the multiplexers select the carry outputs using the previously generated carry outputs. In particular, the carry output C_0 in 4-bit CCA is selected from C_0^0, C_0^1 using C_{in} as select signal as shown in Fig. 6. The XOR gate is used to generate the sum outputs using S^* and corresponding carry output. Next, we propose a modified half adder in IMPLY logic.

A. Modified Half Adder

We propose an IMPLY-based 1-bit adder called as *modified half adder* (MHA), which has two inputs (namely a and b) and generates one sum (S^*) and two carry outputs (namely, C^0 and C^1). The modified half adder generates the sum (S^*) of a, b as $S^* = a \oplus b$. While the carry outputs are generated considering C_{in} as '0' and '1' as $C^0 = a\&b$ and $C^1 = a + b$ respectively. IMPLY-based MHA is shown in Fig. 7(a), which takes four memristors and eleven computational steps as listed in Table VI. The carry outputs C^0 and C^1 generated in steps 4 and 6 respectively while SUM (S^*) is available in step 11 and stored in the memristors $M1, B, M2$ respectively.

In a 4-bit CCA, the outputs S_0 and C_0 are the sum and carry of a_0, b_0 and C_{in}, generated using a full adder, which uses an MHA and a MUX as shown in Fig. 6. The full adder takes sixteen computational steps and requires seven memristors. While the final carry is calculated using XOR gate as shown in Fig. 6. Fig. 7(b) depicts the IMPLY-based circuit for XOR and Table VII gives the IMPLY operations for XOR. All the XOR operations for sum are performed simultaneously in 7 steps using two memristors.

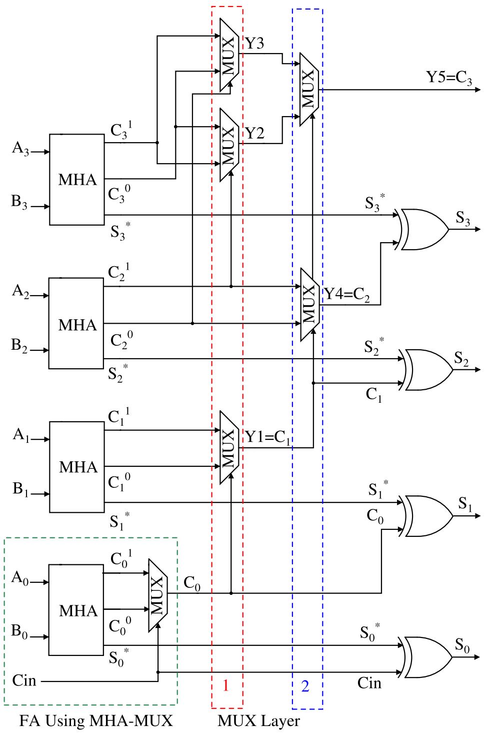

Fig. 6. Proposed 4-bit Conditional Carry Adder.

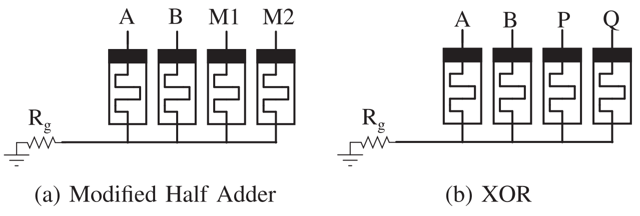

Fig. 7. Conditional Carry Adder circuit elements in IMPLY logic.

TABLE VI
IMPLY Logic Steps for Proposed Modified Half Adder

Steps	IMPLY Operation	Equivalent Logic
1	$A \rightarrow M1$	$M1 = \overline{A}$
2	$B \rightarrow M2$	$M2 = \overline{B}$
3	$A \rightarrow M2$	$M2 = A \rightarrow M2 = \overline{A} + \overline{B}$
4	$M1 \rightarrow B$	$B = M1 \rightarrow B = A + B = \mathbf{C^1}$
5	$FALSE(M1)$	$M1 = 0$
6	$M2 \rightarrow M1$	$M1 = \overline{M2} = AB = \mathbf{C^0}$
7	$FALSE(A)$	$A = 0$
8	$B \rightarrow A$	$A = \overline{B} = \overline{A + B}$
9	$M2 \rightarrow A$	$A = \overline{A \oplus B}$
10	$FALSE(M2)$	$M2 = 0$
11	$A \rightarrow M2$	$M2 = \overline{A} = A \oplus B = \mathbf{S^*}$

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
COMPUTATIONAL STEPS FOR PROPOSED XOR LOGIC IN IMPLY LOGIC

Steps	IMPLY Operation	Equivalent Logic
1	$A \rightarrow P$	$P = \overline{A}$
2	$P \rightarrow Q$	$Q = A$
3	$B \rightarrow A$	$A = \overline{B} + A$
4	$Q \rightarrow B$	$B = \overline{A} + B$
5	$FALSE(P)$	$P = 0$
6	$B \rightarrow P$	$P = A\overline{B}$
7	$A \rightarrow P$	$P = B\overline{A} + A\overline{B}$

TABLE IX
EXECUTION OF SUM IN 4-BIT CONDITIONAL CARRY ADDER

SUM Operation			
SUM Output	Switches On	Inputs	Memristors Used
S_0	1	S_0^*, Cin	M2,Cin
S_1	3, 4, 9	S_1^*, C_0	M4,C_0
S_2	6, 21, 5	S_2^*, C_1	M6, Y1
S_3	7, 8, 12, 14, 16	S_3^*, C_2	M8, Y4

TABLE X
SUMMARY OF IMPLY STEPS FOR 4-BIT CCA

Operation	Steps	No. of Memristors
Modified Half Adder	11	16
Full Adder using MHA & MUX	5	3
Copying values for the first layer of MUX	5	13
First layer of MUX execution	5	6
Copying values for the second layer of MUX	3	3
Second layer of MUX execution	5	4
Sum S_3, S_2, S_1, S_0	7	8
Total	41	49

TABLE XI
NO. OF COPY OPERATIONS IN THE FIRST LAYER OF MUX IN n-BIT CCA

Copy	CCA size					
	4	8	16	32	64	n
One time	5	10	19	36	69	$n + \log(n/2)$
Two time	1	3	7	15	31	$\frac{n}{2} - 1$
Three time	0	1	4	11	26	$\frac{n}{2} - \log n$

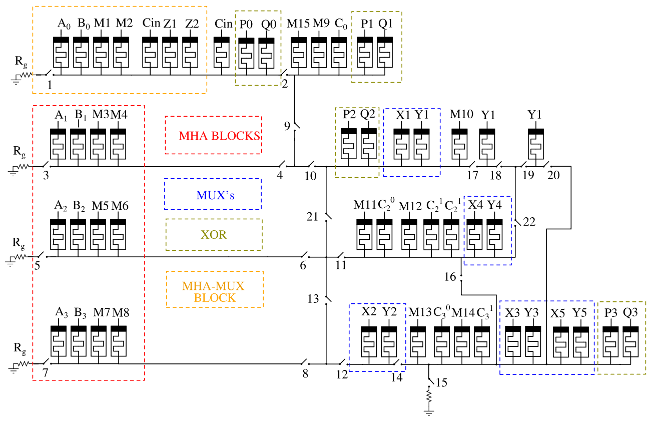

Fig. 8. Proposed IMPLY based 4-bit Conditional Carry Adder.

TABLE VIII
MUX EXECUTION IN 4-BIT CONDITIONAL CARRY ADDER

First Layer MUX Operation			
MUX	Switches On	Inputs	Memristors Used
1	2, 9, 4, 10, 3	C_0, C_1^0, C_1^1	M1, M3, B_1
2	6, 13, 8, 12, 7	C_3^1, C_3^0, C_2^0	B_3, M7, M5
3	15, 16	C_3^1, C_3^0, C_2^1	C_3^1, C_3^0, C_2^0
Second Layer MUX Operation			
MUX	Switches On	Inputs	Memristors Used
4	5, 6, 11, 18	$C_2^0, C_2^1, Y1$	$C_2^0, C_2^1, Y1$
5	20, 14, 15	$Y3, Y2, Y1$	$Y3, Y2, Y1$

B. *Proposed 4-Bit Conditional Carry Adder*

Fig. 8 depicts the memristor-based realization of the IMPLY-based 4-bit CCA. The proposed 4-bit CCA requires 5 multiplexers (excluding the MUX used in full adder) and these multiplexers are categorized in two layers as shown in Fig. 6. In the first layer, three multiplexer operations are performed to generate $Y1 = C_1$ from C_1^0, C_1^1 using C_0 and $Y2, Y3$ from C_3^0, C_3^1 using C_2^0, C_2^1 respectively. The first layer of MUX takes five computational steps using two additional memristors per MUX (as indicated in Fig. 8). The output of the first layer of the multiplexers are stored in the memristors Y1, Y2 and Y3. In the second layer, outputs $Y4$ and $Y5$ are calculated. The multiplexer operations are summarized in Table VIII. These parallel multiplexer operations require some copying operations. In the first layer of multiplexers, $C_0, C_2^0, C_2^1, C_3^0, C_3^1, Cin$ values are copied and $Y1$ is copied in second layer. These copying operations are performed on independent rows as shown in Fig. 8.

Hence, the operations were completed in five computational steps with thirteen memristors in the first layer and three computational steps with three memristors in the second layer. Table IX summarizes the switches used and the memristors involved in generating *sum* output. The sum outputs S_0, S_1, S_2 and S_3 are stored in memristors $P0, P1, P2$ and $P3$, respectively.

Table X summarizes the number of computational steps for the operations involved in 4-bit CCA. A 4-bit CCA takes 41 computational steps and requires 49 memristors.

C. *n-Bit Conditional Carry Adder*

Fig. 9 depicts the 8-bit conditional carry adder, which requires three multiplexer layers. A n-bit CCA requires $\log n$ number of multiplexer layers and n number of adders ('n' number of modified half adders with one MUX). In n-bit CCA, 'n' number of MHA blocks take $4n$ memristors and 11 computational steps. After MHA computations, n-bit CCA performs a few copying operations followed by the selection of carry outputs using the MUX layers.

In the first layer of MUX, even-numbered carry outputs select odd-numbered carry outputs, such as C_2^0 selects one of C_3^0 and C_3^1. Similarly, C_2^1 selects from the C_3^0 and C_3^1. Therefore, C_3^0 and C_3^1 are copied once. From Fig. 9, it is evident to note that the intermediate values $Cin, C_0, C_2^0, C_4^0, C_3^0, C_3^1, C_5^0, C_5^1, C_7^0$ and C_7^1 are copied once. While some intermediate carry outputs need to be copied more than once. In particular, in 8-bit CCA, (C_2^1, C_4^1, C_6^0) are copied twice whereas C_6^1 is copied thrice. Table XI gives the number of copy operations in the first layer of a n-bit CCA.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

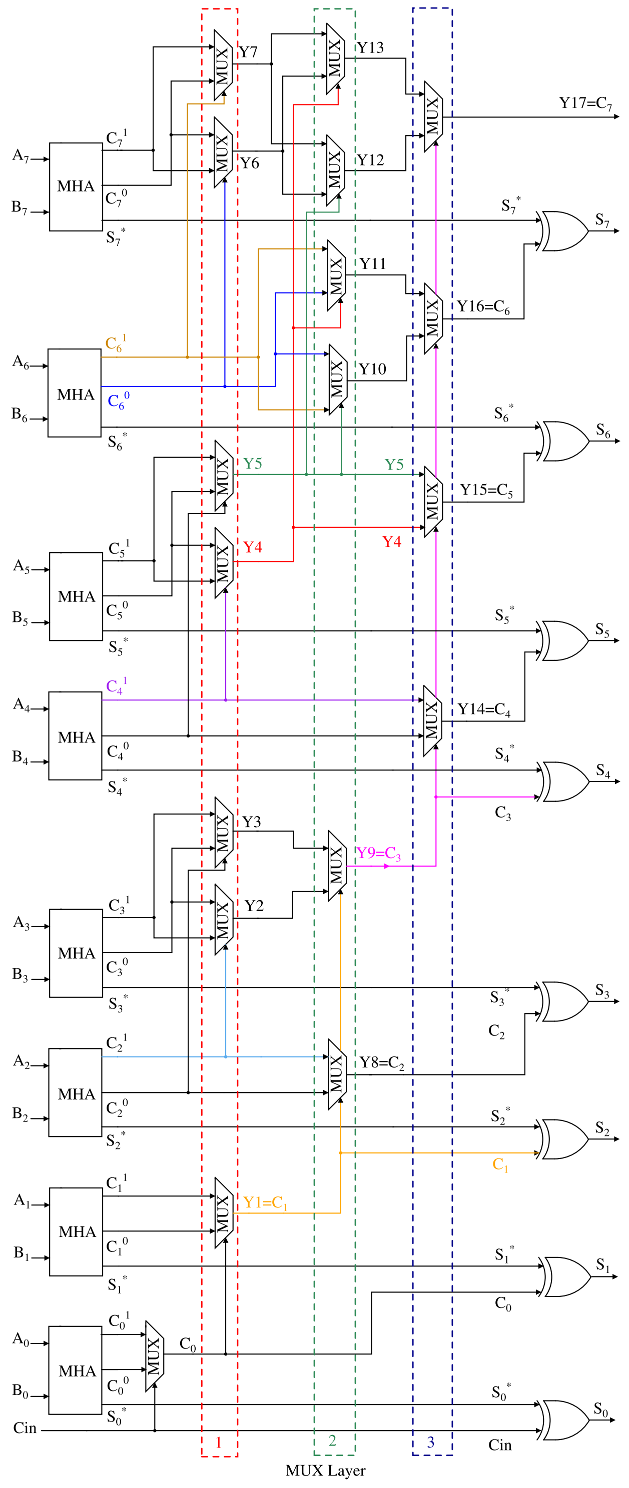

Fig. 9. 8-bit Conditional Carry Adder.

TABLE XII
NO. OF COPY OPERATIONS IN THE SECOND LAYER OF MUX

Copy	CCA size					
	4	8	16	32	64	n
One time	0	2	6	14	30	$\frac{n}{2}-2$
Two time	1	3	5	7	9	$2\log(n/4)+1$
Three time	0	0	2	8	24	$f(n)$

TABLE XIII
NO. OF COPY OPERATIONS IN VARIOUS LAYERS OF MUX IN n-BIT CCA

No. of Copy Operations	CCA size					
	4	8	16	32	64	n
3rd layer of MUXs						
4	0	1	3	7	15	$\frac{n}{4}-1$
4th layer of MUXs						
8	0	0	1	3	7	$\frac{n}{8}-1$
5th layer of MUXs						
16	0	0	0	1	3	$\frac{n}{16}-1$

TABLE XIV
NUMBER OF MEMRISTORS FOR COPY OPERATIONS IN n-BIT CCA

MUX Layer	Number of times a value is copied						
	1	2	3	4	8	16	32
First	$n+\log(n/2)$	$\frac{n}{2}-1$	$\frac{n}{2}-\log(n)$	-	-	-	-
Second	$\frac{n}{2}-2$	$2\log(n/4)$ $+1$	$f(n)$	-	-	-	-
Third	-	-	-	$\frac{n}{4}-1$	-	-	-
Fourth	-	-	-	-	$\frac{n}{8}-1$	-	-
Fifth	-	-	-	-	-	$\frac{n}{16}-1$	-
Sixth	-	-	-	-	-	-	$\frac{n}{32}-1$
No. of memristor for copy	2	3	4	5	9	17	33
Total memristors	$3n-4$ $+2\log(\frac{n}{2})$	$\frac{3n}{2}$ $+6\log(\frac{n}{4})$	$2n-4\log(n)$ $+\,4f(n)$	$5(\frac{n}{4}-1)$	$9(\frac{n}{8}-1)$	$17(\frac{n}{16}-1)$	$33(\frac{n}{32}-1)$

TABLE XV
NO. OF COMPUTATIONAL STEPS FOR COPY OPERATIONS IN n-BIT CCA

MUX Layer	$n=4$	$n=8$	$n=16$	$n=32$	$n=64$
First	5	7	7	7	7
Second	3	4	4	4	4
Third	—	5	5	5	5
Fourth	—	—	9	9	9
Fifth	—	—	—	17	17
Sixth	—	—	—	—	32

In the second layer of multiplexers, some signals were copied once, twice and thrice similar to first layer. For instance, in Fig. 9, the values $Y6$ and $Y7$ are copied once while the values $Y1, Y4, Y5$ are copied twice. Table XII gives required copy operations to execute the second layer of the MUX, here $f(n)$ in Table XII is defined via (5).

$$f(n) = \begin{cases} 0, & \text{if } n = 4, 8 \\ 2, & \text{if } n = 16 \\ \frac{n}{2}\left(1 - \frac{1}{2^{\frac{n}{32}}}\right), & \text{if } n \geq 32 \end{cases} \tag{5}$$

In 8-bit CCA, while executing the third layer of MUX, $Y9$ is used five times, hence the same is copied four times. In n-bit CCA, from the third layer onwards the number of copy operations increase with n as 2^{m-1}, here m is layer number defined as $m = 3, 4, \ldots, \log n$. The number of copy operations are given in Table XIII. Table XIV shows the number of memristors required for each copy operation in various layers of multiplexers. Total number of memristors can be calculated by adding the required memristors for operations such as MHA, MUX, XOR, and copy.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

TABLE XVI
NO. OF MEMRISTORS & COMPUTATIONAL STEPS FOR n-BIT CCA

Component	No. of Memristors	No. of Steps
MHA and Full adder	$4n + 3$	16
Copy Operations	Table XIV	Table XV
MUX's	$2(\sum_{k=1}^{\log_2 n} 2^{\log_2 n} - 2^{(\log_2 n)-k})$	$5\log n$
XOR's	$2n$	7

TABLE XVII
NO. OF MEMRISTORS AND STEPS FOR 32-BIT CCA

Operation	Number of Memristors	Number of steps
MHA and Full adder	$(4*32)+3 = 131$	16
Copy	305 from Table XIII	$7+4+5+9+17 =$ 42 from Table XV
MUXs	$2*(31+30+28+24+16) = 258$	$5*\log 32 = 25$
XOR	$2*32 = 64$	7
Total	758	90

Table XVI gives the computational steps for operations involved in n-bit CCA. For example, 8-bit CCA requires 11 computational steps for MHA followed by 5 steps for MUX in full adder. The copy operations are as follows. The carry outputs $Cin, C_0, C_2^0, C_3^0, C_3^1, C_4^0, C_5^0, C_5^1, C_7^0, C_7^1$ are copied once as shown in Fig. 9. C_2^1, C_4^1, C_6^0 were duplicated twice while C_6^1 copied thrice. Since the proposed architecture has independent rows for copy operations, some copy operations can be performed in parallel. While C_6^0, C_6^1 lies on the same row, it requires 7 IMPLY steps for copying. Hence, copy takes 7 computational steps. Then, the first layer of MUX is executed simultaneously in 5 computational steps. From Fig. 9, values $Y6$ and $Y7$ are copied once and $Y1, Y4, Y5$ are copied twice for the execution of second layer of MUX. The copy operations are executed concurrently in 4 steps to prepare for the parallel execution of the MUXs in the second layer in 5 steps. Value Y9 $= C_3$ is copied four times in 5 computational steps, followed by another 5 computational steps for the third layer of MUX execution. Finally, the sum outputs were generated simultaneously in 7 steps. Hence, it takes 54 computational steps to compute 8-bit CCA in IMPLY logic. Table XVI lists the number of steps needed for copying for various n-bit CCA.

Table XVI shows the number of computational steps and memristors for each phase and altogether gives the total number of computational steps and requried number of memristors. In particular, IMPLY-based 32-bit CCA takes 90 computational steps with 758 memristors as shown in Table XVII. The advantages of the proposed CCA given via Remark 2.

Remark 2: The proposed CCA takes less computational steps compared to carry select and ripple carry adders because of proposed MHA and the inherent parallel computations. In particular, proposed 32-bit CCA requires 90 computational steps while the proposed CSA takes 123 steps for the same addition. Whereas, a ripple carry adder presented in [25] takes 176 steps.

TABLE XVIII
VTEAM [29] MODEL PARAMETERS

Parameter	Value	Parameter	Value
k_{on}	$-216.2\,m/sec$	k_{off}	$0.091\,m/sec$
R_{on}	$1\,K\Omega$	R_{off}	$300\,K\Omega$
$V_{T,on}$	$-1.5\,V$	$V_{T,off}$	$0.3\,V$
x_{on}	0	x_{off}	$3\,nm$
α_{on}	4	α_{off}	4

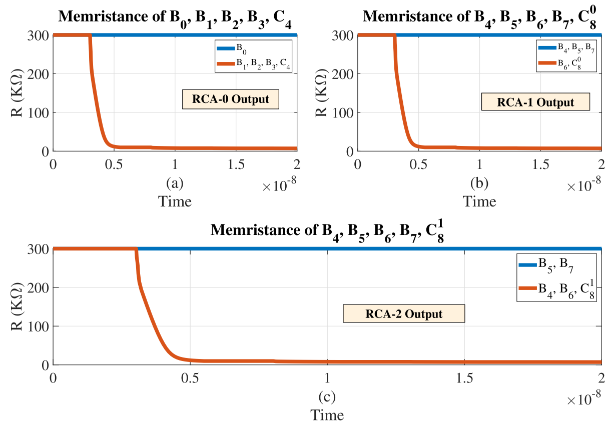

Fig. 10. RCA outputs after 36 IMPLY steps (a) RCA-0: $S_{3:0} =$ "1110" and $C_4 =$ '1'; (b) RCA-1 for $Cin = 0$: $S_{7:4}^0 =$ "0100" and $C_8^0 =$ '1'; (c) RCA-2 for $Cin = 1$: $S_{7:4}^1 =$ "0101" and $C_8^1 =$ '1'.

V. SIMULATION RESULTS

Proposed designs are simulated in Cadence Virtuoso using VTEAM model [29]. Among the resistance switching of memristive materials such as *Pt-Hf-Ti* based memristors [30], Ferroelectric memristors [31] and metallic nanowire memristors [32], the VTEAM model is accurate enough, displaying less than 1.5% relative root mean square error [33]. The parameters of the VTEAM model used in the simulations are listed in Table XVIII. These parameters are chosen to produce a delay of 1.05 ns for a single memristor SET and RESET operations which is an identical delay to fit into the real-world device as reported in [34]. The IMPLY parameters used for the simulations are as follows, $\{V_{SET}, V_{COND}, V_{RESET}, R_g, t_{pulse}\}$ $=\{2\,V, 1.5\,V, -1\,V, 2\,K\Omega, 5\,ns\}$.

A. Simulations of 8-Bit Carry Select Adder

Simulations for 8-bit carry select adder are performed by considering the inputs $A =$ "10101111", $B =$ "10101111" and initial $Cin = 0$. Fig. 10 shows the *RCA-0*, *RCA-1* and *RCA-2* outputs after execution of 36 computational steps. Memristors B_0, B_1, B_2, B_3 stores the output S_0, S_1, S_2, S_3 respectively whereas memristor C stores C_4. From Fig. 10(a) values stored in B_0, B_1, B_2, B_3 are '0, 1, 1, 1' indicating $S_0 =$ '0', $S_1 =$ '1', $S_2 =$ '1' and $S_3 =$ '1'. Similarly, Fig. 10(b) and (c) gives the outputs of *RCA-1* and *RCA-2* under $Cin =$ '0' and $Cin =$ '1' assumption.

Fig. 11(a) and (b) gives the simulation of the MUX output after 46 and 51 IMPLY steps respectively. Memristors $Y1, Y2, Y3, Y4$ store the values of S_4, S_5, S_6, S_7 respectively

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

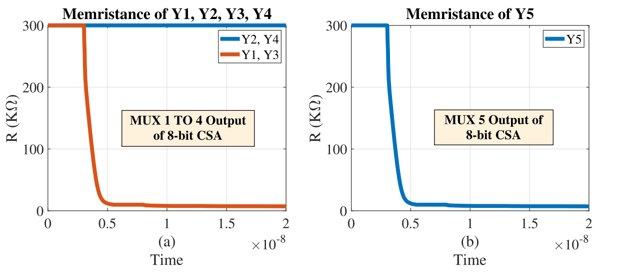

Fig. 11. Y1 to Y5 have the outputs of MUX-1,2,3,4 and 5; (a) After 46 IMPLY steps, $S_{7:4}$ = "0101" and (b) After 51 IMPLY steps C_8 = '1'.

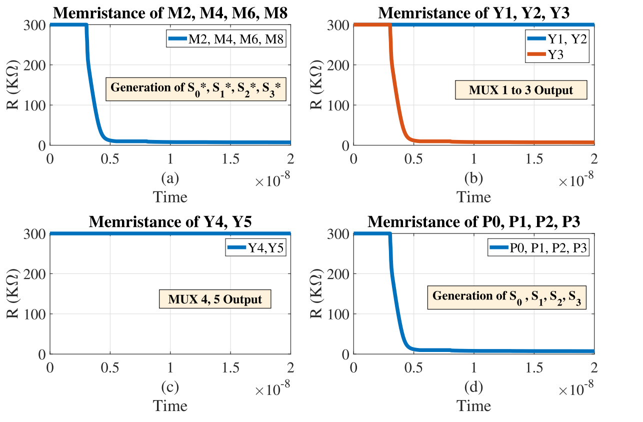

Fig. 12. (a) M2, M4, M6, M8 stores MHA sum after 11 IMPLY steps, indicating $S^*_{3:0}$ ="1111"; (b) Output of MUX-1,2 and MUX-3, as $Y1$ = '0', $Y2$ = '0' and $Y3$ = '1'; (c) Outputs of MUX 4, 5 for C_2, C_3 as '0' and (d) P0, P1, P2, P3 stores sum in 7 IMPLY steps as $S_0 = S_1 = S_2 = S_3$ = '1'.

indicating the values as S_4 = '1', S_5 = '0', S_6 = '1' and S_7 = '0'. Final C_8 is stored in memristor $Y5$ as shown in Fig. 11(b) indicating C_8 = '1'. It is evident to note from Figs. 10 and 11, the sum of "10101111" and "10101111" for C_{in} = '0' is "01011110", with '1' as carry output.

B. Simulations of 4-Bit Conditional Carry Adder

Simulations were carried out for 4-bit CCA considering A = "1010", B = "0101" and Cin = '0'. Fig. 12(a) shows the memristance of $M2, M4, M6, M8$ memristors storing the MHA outputs after 11 IMPLY steps. Fig. 12(b) shows the memristance of $Y1, Y2, Y3$ as the output of MUX-1,2,3 as '0", '0", '1". Fig. 12(c) shows the outputs of MUX-4,5 for C_2 and C_3. After the final XOR operation for sum computation, *P0, P1, P2, P3* shown in Fig. 12(d) have the sum S_0, S_1, S_2, S_3 as '1'. It is evident to note that the sum of "1010" and "0101" for C_{in} = '0' is "1111" with '0' as carry.

Next, we will discuss the comparisons of the proposed adders with the existing designs.

VI. Comparisons

This work is centered on improving the addition speed by proposing two alternative adder designs compared to the traditional RCA design in IMPLY logic. As no existing IMPLY-based CSA and CCA designs are available in the literature for a

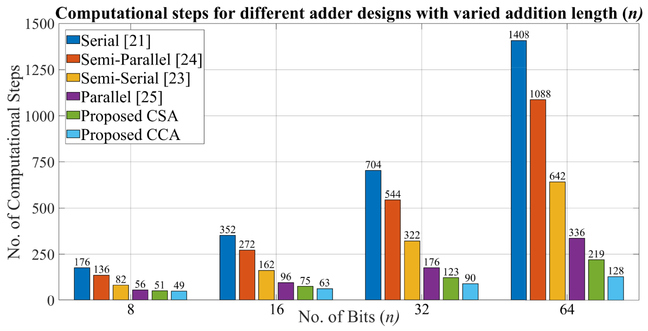

Fig. 13. No. of computational steps for various adders.

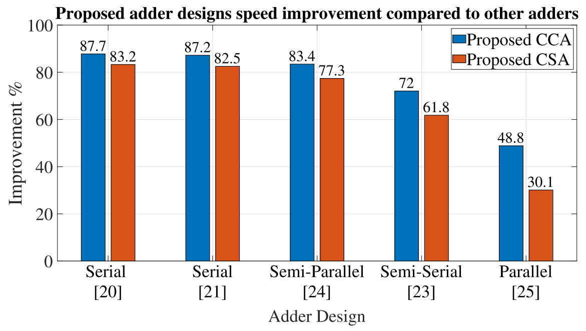

Fig. 14. Comparison of speed improvement of proposed adders with existing 32-bit adders.

similar comparison, the proposed designs are evaluated against the current IMPLY-based RCA designs. The performance of memristor-based arithmetic circuits can be validated using various parameters, such as speed (measured in the number of computational steps), area (measured in the number of memristors), and energy consumption. A *Figure of Merit* (FoM) is introduced in [35] for evaluating the memristor designs based on area and speed constraints. Different FoMs are proposed in [35] for balanced, area-focused, or speed-focused scenarios. A speed-focused FoM (FoM_S) gives higher priority to the computational steps and is defined as $1/(f * g^2)$, where f is the number of memristors, and g is the number of computational steps. While the balanced FoM (FoM_B) is employed when both area and speed are of equal importance and is defined as $1/(f * g)$.

Table XIX provides a comparative analysis of the proposed adders and existing designs based on the number of memristors, computational steps, and energy consumption. From Table XIX, it is noted that the proposed adders exhibit the lowest number of computational steps among all other adders. Fig. 13 compares the computational steps for 8, 16, 32, and 64-bit adders. The proposed adder designs exhibit greater efficiency in terms of computational steps compared to existing designs. Fig. 14 illustrates the percentage of improvement in the proposed adder's speed compared to the existing adders. Specifically, the proposed CSA and CCA designs demonstrate a speed improvement of 82% and 87%, respectively, compared to the serial design presented in [21]. Furthermore, the proposed CSA and CCA

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

TABLE XIX
COMPARISON OF PROPOSED ADDERS WITH EXISTING DESIGNS

Design \ Size	Energy Consumption (nJ)				No. of Memristors					No. of Steps					FoM_S (μ)	FoM_B (μ)
	4-bit	8-bit	16-bit	32-bit	4-bit	8-bit	16-bit	32-bit	n-bit	4-bit	8-bit	16-bit	32-bit	n-bit	$n = 32$	$n = 32$
Serial [20]	0.56	1.11	2.23	4.47	11	19	35	67	$2n + 3$	11	19	35	67	$23n$	0.0275	20.2
Serial [21]	0.53	1.07	2.14	4.2	11	19	35	67	$2n + 3$	88	176	352	704	$22n$	0.0301	21.2
Semi-Serial [23]	0.48	0.97	1.9	3.8	14	22	38	70	$2n + 6$	42	82	162	322	$10n + 2$	0.137	44.3
Semi-Parallel [24]	0.53	1.07	2.14	4.2	11	19	35	67	$2n + 3$	68	136	272	544	$17n$	0.0504	27.4
Parallel [25]	0.51	1.02	2.04	4.08	17	33	64	129	$4n + 1$	36	56	95	176	$5n + 16$	0.25	44.04
Proposed CSA	0.87	1.71	3.41	6.78	44	82	158	310	$19\frac{n}{2} + 6$	39	51	75	123	$3n + 27$	0.213	26.22
Proposed CCA	0.64	1.48	3.51	7.71	49	136	331	758	Table XVI	41	54	68	90	Table XVI	0.189	14.6

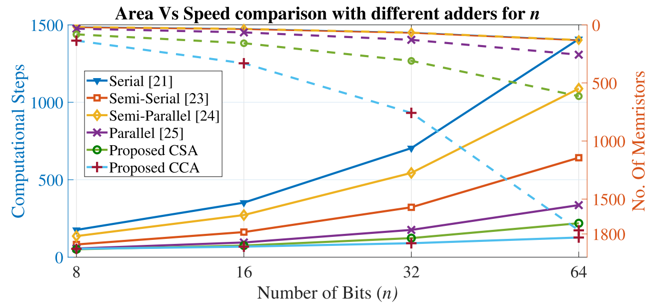

Fig. 15. Area and Speed for Various Adders.

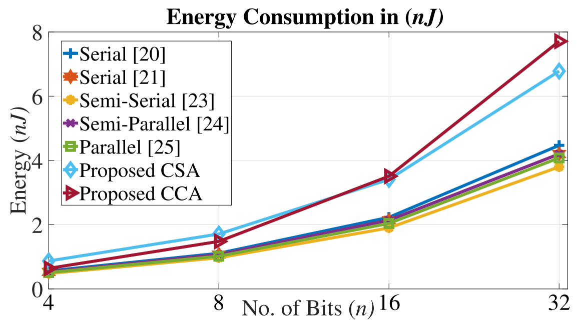

Fig. 16. Comparing energy consumption of proposed adders with other adder designs.

TABLE XX
ENERGY CONSUMPTION OF TWO INPUT IMPLY LOGIC

Input	00	01	10	11
Energy (pJ)	0.691	8.868	4.993	9.772

designs exhibit an improvement of 61% and 72% over the semi-serial [23], 77% and 83% over the semi-parallel adder [24] respectively. Compared to the best existing design [25], the proposed CSA and CCA designs are 30% and 49% faster. Fig. 15 shows the trends of speed and area for various adders and highlights that the proposed adders are high-speed designs compared to existing adders due to their incorporation of a greater number of parallel computations, which requires a higher number of memristors. From Table XIX, the proposed adder designs exhibit higher values of FoM_S compared to existing designs, except for the parallel RCA design. This is due to more memristors required in the proposed designs to facilitate a larger number of simultaneous computations. However, the higher FoM value of our proposed adders implies that they are better suited for applications where high-speed addition is a primary consideration.

Energy Consumption

Energy is calculated based on an IMPLY logic's average switching energy consumption. The energy consumption for all combinations of IMPLY logic is listed in Table XX. The average energy of all the inputs is $6.081\,pJ$. The average energy of IMPLY can be used to calculate overall energy of the proposed adders. For instance, energy estimation for 4-bit CCA can be calculated as follows. The MHA with 11 IMPLY steps consume $66.891\,pJ$, while XOR with 7 IMPLY steps and MUX with 5 IMPLY steps take $42.567\,pJ$ and $30.405\,pJ$ of energy respectively. A copy operation takes $2.842\,pJ$ of energy. Thus a 4-bit CCA, consumes a total of $640\,pJ$. Similarly, a 4-bit CSA with three RCA, six copy, and three MUX consumes $((127.701 * 3) + (6 * 2.842) + (3 * 30.405))\ 874.47\,pJ$ of energy. The reported energy measurements are based on the memristor model, its parameters, the number of cycles, and the pulse width of each cycle. Since the average power consumption for other adder designs has not been provided in the literature, the energy consumption of other designs is calculated assuming the same parameters used for the proposed designs to have a similar comparison. Table XIX describes the energy consumption for proposed adders and Fig. 16 compares the energy consumed by different adders with the proposed adders. The proposed adders consume more energy than the other RCA adder designs because of their complex architecture involving more parallel computations needing more memristors than relatively simple architecture and fewer parallel computations in RCA design.

VII. CONCLUSION

This article presents two high-speed adders such as Carry Select Adder and Conditional Carry Adder in IMPLY logic for in-memory computing through memristive technology. The proposed n-bit carry select adder is designed using $\frac{n}{2}$-bit ripple carry adders and multiplexers. While conditional carry adder is designed using modified half adders and multiplexers. The proposed 1-bit modified half adder computes one sum ($a \oplus b$) and two carry outputs (ab and $a + b$) in eleven computational steps. While the proposed MUX takes five computational steps. The number of computational steps and memristors are derived for n-bit carry select and conditional carry adders. The proposed 32-bit carry select and conditional carry adders are having an improvement of 30% and 49% in terms of the number of computational steps compared to the best existing design. Extensive simulations were performed to calculate the energy consumption for the proposed adders.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.

References

[1] S. Gupta, M. Imani, H. Kaur, and T. S. Rosing, "NNPIM: A processing in-memory architecture for neural network acceleration," *IEEE Trans. Comput.*, vol. 68, no. 9, pp. 1325–1337, Sep. 2019.

[2] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Des. Test*, vol. 34, no. 2, pp. 39–50, Apr. 2017.

[3] European Commission and Directorate-General for the Information Society and Media, and J. Irran et al., "Next generation computing roadmap: Final report," M. Sawyer, A. Stewart, I. Morgan, M. Parsons, and C. Zsigri, Eds., Publications Office, 2014, doi: 10.2759/4587.

[4] S. Khoram, Y. Zha, J. Zhang, and J. Li, "Challenges and opportunities: From near-memory computing to in-memory computing," in *Proc. ACM Int. Symp. Phys. Des.*, 2017, pp. 43–46.

[5] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2013.

[6] H. Sharma et al., "From high-level deep neural models to FPGAs," in *Proc. IEEE/ACM 49th Annu. Int. Symp. Microarchitecture*, 2016, pp. 1–12.

[7] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2017, pp. 481–492.

[8] V. Seshadri et al., "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proc. IEEE/ACM 50th Annu. Int. Symp. Microarchitecture*, 2017, pp. 273–287.

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

[10] Y. Li et al., "A survey of MRAM-centric computing: From near memory to in memory," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 2, pp. 318–330, Apr.–Jun. 2023.

[11] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "The desired memristor for circuit designers," *IEEE Circuits Syst. Mag.*, vol. 13, no. 2, pp. 17–22, Secondquarter 2013.

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

[13] J. Borghetti et al., "'memristive'switches enable 'stateful'logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[14] J. Sun, G. Han, Z. Zeng, and Y. Wang, "Memristor-based neural network circuit of full-function pavlov associative memory with time delay and variable learning rate," *IEEE Trans. Cybern.*, vol. 50, no. 7, pp. 2935–2945, Jul. 2020.

[15] J. Sun, Y. Wang, P. Liu, S. Wen, and Y. Wang, "Memristor-based neural network circuit with multimode generalization and differentiation on pavlov associative memory," *IEEE Trans. Cybern.*, vol. 53, no. 5, pp. 3351–3362, May 2023.

[16] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL–Memristor Ratioed Logic," in *Proc. IEEE 13th Int. Workshop Cellular Nanoscale Netw. their Appl.*, 2012, pp. 1–6.

[17] D. B. Strukov et al., "Hybrid CMOS/memristor circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 1967–1970.

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

[19] 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.

[20] 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. IEEE 21st Int. Conf. Electron., Circuits Syst.*, 2014, pp. 562–565.

[21] S. G. Rohani and N. TaheriNejad, "An improved algorithm for imply logic based memristive full-adder," in *Proc. IEEE 30th Can. Conf. Elect. Comput. Eng.*, 2017, pp. 1–4.

[22] 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.*, 2017, pp. 695–698.

[23] N. Taherinejad, T. Delaroche, D. Radakovits, and S. Mirabbasi, "A semi-serial topology for compact and fast imply-based memristive full adders," in *Proc. IEEE 17th Int. New Circuits Syst. Conf.*, 2019, pp. 1–4.

[24] S. G. Rohani, N. Taherinejad, and D. Radakovits, "A semiparallel full-adder in IMPLY logic," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 28, no. 1, pp. 297–301, Jan. 2020.

[25] A. Karimi and A. Rezai, "Novel design for a memristor-based full adder using a new imply logic approach," *J. Comput. Electron.*, vol. 17, no. 3, pp. 1303–1314, 2018.

[26] O. J. Bedrij, "Carry-select adder," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 3, pp. 340–346, 1962.

[27] J. Sklansky, "Conditional-sum addition logic," *IRE Trans. Electron. Comput.*, vol. EC-9, no. 2, pp. 226–231, Jun. 1960.

[28] K.-H. Cheng, S.-M. Chiang, and S.-W. Cheng, "The improvement of conditional sum adder for low power applications," in *Proc. IEEE 11th Annu. Int. ASIC Conf.*, 1998, pp. 131–134.

[29] 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.

[30] E. Yalon et al., "Resistive switching in HfO_2 probed by a metal–insulator–semiconductor bipolar transistor," *IEEE Electron Device Lett.*, vol. 33, no. 1, pp. 11–13, Jan. 2012.

[31] A. Chanthbouala et al., "A ferroelectric memristor," *Nature Mater.*, vol. 11, no. 10, pp. 860–864, 2012.

[32] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE Electron Device Lett.*, vol. 32, no. 10, pp. 1436–1438, Oct. 2011.

[33] 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.

[34] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 1, pp. 13–24, 2013.

[35] D. Radakovits, N. TaheriNejad, M. Cai, T. Delaroche, and S. Mirabbasi, "A memristive multiplier using semi-serial imply-based adder," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 67, no. 5, pp. 1495–1506, May 2020.

Nandit Kaushik (Graduate Student Member, IEEE) received the B.Tech. degree in electronics and communication engineering from DCRUST, Sonipat, India, in 2016 and the M.Tech. degree in electronics and communication engineering from The LNM Institute of Information and Technology, Jaipur, India, in 2019. He is currently working toward the Doctoral degree with the School of Computing and Electrical Engineering, Indian Institute of Technology Mandi, Mandi, India. His research interests include developing memristor-based neuromorphic architectures for In-memory-computing and their efficient implementation inside a memristive crossbar.

Srinivasu Bodapati (Member, IEEE) received the Ph.D. degree in VLSI from the Department of Electrical Engineering, Indian Institute of Technology Madras, Madras, India, in 2017. From 2017 to 2019, he was a Research Fellow with Nanyang Technological University, Singapore. Since 2019, he has been an Assistant Professor with the School of Computing and Electrical Engineering, Indian Institute of Technology Mandi, Mandi, India. His research interests include digital design with emerging device technologies, in memory computing architectures using memristive devices, field-programmable gate array (FPGA)-based designs, and FPGA implementation of cryptographic primitives.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on December 31,2024 at 19:02:02 UTC from IEEE Xplore. Restrictions apply.