Accelerated Image Processing Through IMPLY-Based NoCarry Approximated Adders

Fabian Seiler and Nima TaheriNejad, *Member, IEEE*

***Abstract*— As the demand for computational power increases drastically, traditional solutions to address those needs struggle to keep up. Consequently, there has been a proliferation of alternative computing paradigms aimed at tackling this disparity. Approximate Computing (AxC) has emerged as a modern way of improving speed, area efficiency, and energy consumption in error-resilient applications such as image processing or machine learning. The trade-off for these enhancements is the loss in accuracy. From a technology point of view, memristors have garnered significant attention due to their low power consumption and inherent non-volatility that makes them suitable for In-Memory Computation (IMC). Another computing paradigm that has risen to tackle the aforementioned disparity between the demand growth and performance improvement. In this work, we leverage a memristive stateful in-memory logic, namely Material Implication (IMPLY). We investigate advanced adder topologies within the context of AxC, aiming to combine the strengths of both of these novel computing paradigms. We present two approximated algorithms for each IMPLY based adder topology. When embedded in an Ripple Carry Adder (RCA), they reduce the number of steps by 6% − 54% and the energy consumption by 7% − 54% compared to the corresponding exact full adders. We compare our work to State-of-the-Art (SoA) approximations at circuit-level, which improves the speed and energy efficiency by up to 72% and 34%, while lowering the Normalized Median Error Distance (NMED) by up to 81%. We evaluate our adders in four common image processing applications, for which we introduce two new test datasets as well. When applied to image processing, our proposed adders can reduce the number of steps by up to 60% and the energy consumption by up to 57%, while also improving the quality metrics over the SoA in most cases.**

***Index Terms*— Approximate, memristor, in-memory computing, IMPLY, energy efficiency, image processing.**

I. Introduction

ADDITION operations serve as foundational components in digital arithmetic, as a significant portion of basic instructions depends on both addition and multiplication [1]. To meet the rapidly escalating demand for processing power, improving adders is a pivotal factor in enhancing overall computing performance. With the slowdown of Moore's Law [2] and transistors reaching their physical limits [3], and the exponentially increasing larger footprint of computing [4], attention is increasingly focused on exploring new computing paradigms and emerging technologies.

AxC emerges rapidly as a promising solution to enhance computing performance and address the power-wall problem [2], [5]. By approximating computational processes, significant enhancements in speed, area, and energy consumption can be achieved. The accuracy is reduced as a trade-off for these improvements [1], [2], [5]. Due to the error-resilient nature of image and video processing applications, approximating certain aspects can significantly improve computing time and power consumption [2], [6], [7]. Additionally, relevant fields such as machine learning, pattern recognition, communication, data mining, and robotics, which are closely intertwined with imaging applications, would also benefit [2], [8], [9], [10], [11].

Several approximated adders based on Complementary Metal-Oxide Semiconductor (CMOS) technology have been introduced [1], [12], [13], [14]. They all share the underlying problem of the Von-Neumann bottleneck that typically occurs between logic and memory. IMC represents an approach for performing computations directly within memory, offering a potential solution to this issue. The memristor [15] stands out as a promising emerging component. With its inherent capability to store data non-volatile through its resistive state and perform logical operations, the memristor is the ideal candidate for IMC memory cells [16], [17]. Additional attributes, such as low power consumption and compact form factor, further position memristors as promising candidates for future computing advancements [18], [19], [20], [21]. In the context of IMC, the stateful logic IMPLY emerges as one of the most popular choices. It is compatible with the crossbar array, and it turned out to be the most reliable stateful logic in [22]. Deeming it an optimal candidate for such applications [16], [23]. The structures currently available for performing IMPLY operations can be categorized into serial, parallel, and hybrid topologies [19], [24], [25], [26]. Each topological implementation offers distinct advantages in one or more metrics such as speed or area usage, rendering them competitive.

In this work, we present two approximated IMPLY-based adders for the serial, parallel, semi-serial, and semi-parallel

Manuscript received 28 April 2024; revised 13 June 2024 and 6 July 2024; accepted 7 July 2024. This article was recommended by Associate Editor X. Lou. *(Corresponding author: Nima TaheriNejad.)*

Fabian Seiler is with the Faculty of Electrical Engineering and Information Technology, Institute of Computer Technology (ICT), Technische Universität Wien (TU Wien), 1040 Vienna, Austria (e-mail: fabian.seiler@student.tuwien.ac.at).

Nima TaheriNejad is with the Faculty of Electrical Engineering and Information Technology, Institute of Computer Technology (ICT), Technische Universitat Wien (TU Wien), 1040 Vienna, Austria, and also with the Faculty of Engineering, Institute of Computer Engineering (ZITI), Heidelberg University, 69117 Heidelberg, Germany (e-mail: nima.taherinejad@ziti.uni-heidelberg.de).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2024.3426926.

Digital Object Identifier 10.1109/TCSI.2024.3426926

topologies. Our proposed methods significantly enhance speed and energy efficiency across all adder structures compared to the exact algorithms. They also outperform SoA approximated adders in both metrics. The key contributions of this paper are:

- Two new approximation methods for designing IMPLY-based adders for each of the four IMPLY-based topologies, leading to eight new approximated adders, thereby, improving the speed, energy consumption, and error metrics compared to SoA exact and approximated adder;
- Introducing approximated adders to the parallel and semi-parallel topology for the first time;
- Proposing a new evaluation dataset for image addition and grayscale filtering;

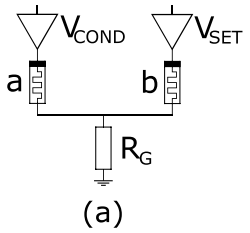
This work is divided into eight sections. In Section II, we will review the literature and cover relevant papers in related areas. The design methodology and the implementation in the different topologies are described in Section III. In Section IV we simulated and verified the adders at circuit-level. The evaluation of standard error metrics can be found in Section V. We compared to other exact and approximated adders in Section VI. The results of three image-processing applications and their quality evaluation can be seen in Section VII, where we also discuss the potential gains. In Section VIII we conclude the paper.

II. Related Work

A. Memristors and IMPLY

The memristor is a novel two-terminal component that was discovered by Chua [15] and physically realized by Strukov et al. [18]. It is a non-volatile memory that can store data with its resistive states. With other advantages such as low power consumption, fast write time, and small dimensions the memristor stands out as an optimal device for a memory cell [16], [17], [19], [20], [21]. The memristor's minimum (R_{on}) and maximum (R_{off}) resistance values are determined by the applied voltage and current direction. They form a hysteresis curve, where R_{on} conventionally corresponds to logical '1' and R_{off} to logical '0' [27], [28], [29].

IMPLY is a stateful logic that is compatible with inherent properties of memristors and has the advantage that no reads and writes are required to perform logical operations [20]. It was introduced by Hewlett Packard (HP), it is compatible with the crossbar array, and it has established itself as a promising candidate for IMC [16], [26], [30]. There exist other stateful logic forms for memristors such as FELIX [31], SIXOR [20], MAGIC [32], and TSML [33]. We selected IMPLY over other options, as it is the most reliable in our memristive technology [22] and the only stateful logic with existing approximations [6], [34], [35]. In Figure 1(a) the basic structure for IMPLY operations is shown. It consists of two memristors to which the different voltages V_{COND} and V_{SET} can be applied, and a resistor that is connected to them. For IMPLY operations to be feasible, the resistor must meet the condition $R_{on} \ll R_G \ll R_{off}$. The applied voltages must also satisfy $V_{COND} < V_C < V_{SET}$, where V_C is the threshold voltage of the memristor [16], [19], [26], [30], [36].



a	b	$b' = a \rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

Fig. 1. IMPLY operation [16]: (a) Gate structure, (b) Truth table.

An IMPLY operation is typically denoted by $a \rightarrow b$, where the inputs correspond the the resistive state of the memristors. This operation is controlled by a short pulse of V_{COND} and V_{SET}, where the state of the b-memristor is overwritten by the result [16], [30]. The truth table of this operation is presented in Figure 1(b).

B. IMPLY-Based Full Adders

IMPLY based adders can be divided into three categories: serial, parallel, and hybrid topologies. Hybrid forms, such as semi-serial or semi-parallel, aim to achieve a better balance between speed and area usage. In the following, we briefly review the key properties of each topology and refer the readers to the cited references for further details.

1) Serial: The serial topology consists of memristors in the same row or column of a crossbar array and can only perform either an IMPLY or a FALSE operation per cycle [16], [17], [27]. This structure is shown in Figure 2(a) and requires $22n$ steps and $2n + 3$ memristors for a n-bit calculation [27].

2) Parallel: The parallel topology consists of multiple rows that are not connected, which enables the parallelization of operations [19], [26]. Via switches each row can be connected to a shared c-memristor, which serves for computing and propagating the carry-out. Not all steps can be parallelized due to the dependency on the previous bit. This structure is shown in Figure 2(b) and requires $5n + 16$ steps, $4n + 1$ memristors, and n external switches for an n-bit addition.

3) Semi-Serial: The semi-serial topology is a hybrid structure that consists of two parallel rows with inputs. Both can be connected to the work and carry memristors via switches [25], [28]. This structure, shown in Figure 2(c), requires $10n + 2$ steps, $2n + 6$ memristors, and 12 additional switches for a n-bit addition.

4) Semi-Parallel: The semi-parallel topology consists of two parallel rows, with one input and a work memristor each (c-memristor at the second row) [24]. The two rows can be connected via a switch to perform operations "between sections". This structure can be seen in Figure 2(d) and requires $17n$ steps, $2n + 3$ memristors, and 3 switches for n-bit.

Table I provides an overview of the different topologies, in which memristor usage and computing time are listed.

C. Approximate Computing (AxC)

The core methodology of AxC entails the redefinition of logic by eliminating or changing gates or individual transistors and formulating a new truth table. This leads to an

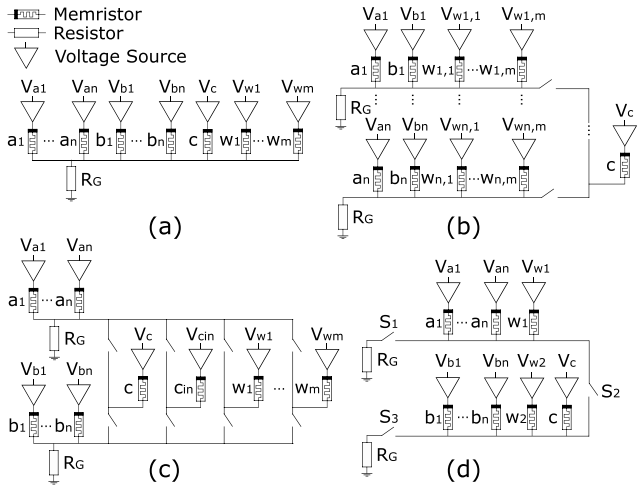


Fig. 2. IMPLY-based topologies: (a) Serial [27], (b) Parallel [19], (c) Semi-Serial [25], (d) Semi-Parallel [24].

TABLE I
COMPARISON OF IMPLY STRUCTURES FOR AN EXACT N-BIT ADDITION

Structure	# of Steps	# of Memristors	# of Switches
Serial [27]	22n	2n+3	0
Parallel [19]	5n+16	4n+1	n
Semi-Serial [28]	10n+2	2n+6	12
Semi-Parallel [24]	17n	2n+3	3

improvement in performance metrics such as energy consumption, area usage, and processing time. The trade-off for this is the accuracy of the calculations [2], [10], [37], [38]. To evaluate the occurring inaccuracy we employ error metrics, namely Error Distance (ED), Error Rate (ER), Mean Error Distance (MED), NMED and Median Relative Error Distance (MRED) [6], [10], [14], [35], [38]. Due to the nature of AxC, only error-resilient applications such as image processing, machine learning, pattern recognition, and robotics are viable [2], [7], [8], [10], [11]. A common quality metric in image processing is the Peak Signal-to-Noise Ratio (PSNR), which represents the noise level. In the SoA a PSNR of over 30dB is considered acceptable [37], [39]. Quality metrics such as Structural Similarity Index Measure (SSIM) and Mean Structural Similarity Index Measure (MSSIM), which represent structural similarity, are employed for assessing image quality. This is done to consider the relevance of structural context for the human visual system [40], [41].

There exist many different approximations in CMOS [5], [12], [14], [42] and other technologies [43], [44]. Recently, there has been a surge in popularity of approximated full adders based on memristors. This trend was initiated by the Memristor Ratioed Logic (MRL)-based approximations introduced in [45] and [46]. IMPLY based approximate full adders in the serial topology have been proposed in [29] and [35]. They simplified the truth table via specific input vectors to reduce the steps and energy consumption by up to 42% and 68%. In [34] approximated adders based on NAND gates are proposed and optimized in the serial structure. In [6] we proposed an approximated adder in the semi-serial topology that reduced the steps and energy consumption by up to 29% and 34%. Here we propose two algorithms for the serial, parallel, semi-serial, and semi-parallel topology. They are significantly faster and more energy efficient than the SoA, while also increasing the image quality in most of the cases.

III. PROPOSED ALGORITHMS

A. Methodology

A major approach in designing approximated circuits is redefining approximate logic functions based on exact logic [2]. This is done by omitting or changing components from the exact circuit or using a modified truth table [2], [38]. As we are working on IMPLY-based topologies, we are only able to use IMPLY and FALSE operations. Since these two functions collectively form the complete logic set $\{\rightarrow, \bot\}$, it is feasible to emulate Boolean logic using only them [27], [47]. An inversion can be emulated with just one IMPLY operation, represented as $\overline{a} = a \rightarrow 0$. This requires a prerequisite of another memristor that has been reset beforehand. OR and NAND are the only functions that can be emulated with two IMPLY operations, making them particularly suitable targets for approximations. The state-of-the-art (SOA) IMPLY-based approximations were developed to reduce the number of steps while keeping the ER for Sum and C_{out} to a minimum [6], [34], [35].

In this study, our aim is not to minimize the ER but rather to concentrate on devising the fastest adder that maintains acceptable quality. Inspired by the approach from [12] we designed and implemented two IMPLY-based algorithms for different adder topologies. We are thereby utilizing the efficient OR emulation that only requires two IMPLY operations. We devised two algorithmic approaches for the serial, semi-serial, parallel, and semi-parallel structure. Throughout the rest of this work, we will denote the algorithms based on their topology combined with IMPLY-based NoCarry (NC) for the base, or NoCarry+ for the advanced implementation. This name was chosen since the carry-out is not propagated (e.g. SINC and SINC+ for the Serial IMPLY-based NoCarry(+) algorithm). The base version only consists of OR combinations between the a and b inputs to set the Sum for each bit. The carry-in is completely disregarded and no carry-out is produced or propagated. So the C_{out} is set to logical '0'. The logical equations for the NoCarry version are

$$C_{out,i}^{NC} = 0 \tag{1}$$

$$Sum_i^{NC} = a_i + b_i = \overline{a_i} \rightarrow b_i. \tag{2}$$

In our second implementation, we advanced the basic version by changing the last approximated bit. We calculate the Sum and C_{out} with the same equations up until the last approximated full adder. At this last bit, we still disregard the carry-in, but produce a carry-out to propagate. To achieve a lower chance of propagating the false C_{out} to the exact bits, this time we set the C_{out} to be a AND b. The logical equations for this extended implementation are

$$C_{out,i}^{NC+} = \begin{cases} 0 & i < k \\ a_i b_i = \overline{a_i \rightarrow \overline{b_i}} & i = k \end{cases} \tag{3}$$

$$Sum_i^{NC+} = a_i + b_i = \overline{a_i} \rightarrow b_i, \tag{4}$$

with k approximated and $n-k$ exact adders embedded in an RCA. For each algorithm proposed, we ensured that the Sum

TABLE II
TRUTH TABLE OF NOCARRY AND NOCARRY+

Inputs			Exact		No Carry		No Carry +	
a	b	c	Sum	Cout	Sum	Cout	Sum	Cout
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	1	0	1	0	1	0
0	1	1	0	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	0	1	0	1	1	0	1	0
1	1	0	0	1	1	0	1	1
1	1	1	1	1	1	0	1	1

TABLE III
REDUCED TRUTH TABLE OF NOCARRY AND NOCARRY+

Inputs			Exact		No Carry		No Carry +	
a	b	c=0	Sum	Cout	Sum	Cout	Sum	Cout
0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	0	0	1	1	0	1	1

TABLE IV
EXACT PROCEDURE OF THE SINC ALGORITHM IN THE SERIAL TOPOLOGY

Steps	Operation	Equivalent Logic
1	$w_1 = 0$	$False(w_1)$
2	$w_1^{'} = a \rightarrow w_1$	$w_1 = \overline{a}$
3	$b^{'} = w_1^{'} \rightarrow b$	$b = \overline{a} \rightarrow b = a + b = Sum$

TABLE V
EXACT PROCEDURE OF THE SINC+ ALGORITHM IN THE SERIAL TOPOLOGY. OPERATIONS COLORED IN BLUE ARE ONLY PERFORMED ONCE AT THE LAST APPROXIMATED BIT OF SINC+

Steps	Operation	Equivalent Logic
1	$w_1 = 0$, $w_2 = 0$	$False(w_1, w_2)$
2	$w_1^{'} = a \rightarrow w_1$	$w_1 = \overline{a}$
-	$w_2^{'} = b \rightarrow w_2$	$w_2 = \overline{b}$
3	$b^{'} = w_1^{'} \rightarrow b$	$b = \overline{a} \rightarrow b = a + b = Sum$
-	$w_2^{''} = a \rightarrow w_2^{'}$	$w_2 = \overline{ab}$
-	$c^{'} = w_2^{''} \rightarrow c$	$c = a \rightarrow \overline{b} = ab = C_{out}$

is stored in the b-memristor, thus guaranteeing compatibility with exact adders. The truth table of both the standard and advanced implementations can be seen in Table II, with the erroneous places marked in red. The ER of Sum is $\frac{4}{8}$ for both implementations, while the ER of C_{out} is $\frac{4}{8}$ for the NoCarry and only $\frac{2}{8}$ for the advanced NoCarry+ adder. Since we want to embed our full adder as the lower bits in an RCA, the carry-in is the C_{out} of the previous bit. Since the C_{out} is set to '0' for all but the last adder in the advanced form, the carry-in is also '0'. With this property of omitted carry propagation, we can reduce the truth table to a form where only a and b can vary and $c = 0$. The reduced truth table can be seen in Table III where the ER of Sum is reduced to $\frac{1}{4}$. The ER of C_{out} is also reduced to $\frac{1}{4}$ for the NoCarry adder, while the NoCarry+ adder has no erroneous C_{out} place at all in the reduced truth table. With this reduced form, only when both a and b are logical '1' an error occurs. It is important to note here that the actual truth table is not reduced. This reduced truth table is used as a visualization technique to highlight which inputs can occur at each approximated bit.

B. Serial Topology: SINC/SINC+

The serial IMPLY topology consists of memristors placed in the same row or column of a crossbar array which are connected to the same resistor [16], [27]. As only one computational area exists, parallelization is not possible, and operations are executed consecutively. The complete algorithm of SINC can be seen in Table IV. In the first step, we reset the work memristor to guarantee the correct functionality. In the second step, we calculate the inversion of a and store it in the work memristor. After that, we save the Sum result, which is $a + b$, in the b-memristor.

The exact procedure of the SINC+ algorithm can be seen in Table V, which has the same approach as the other version. Their difference lies in the operations highlighted in blue, which are only computed only once in the final approximated adder. We therefore need an additional work memristor w_2. It is used to save the inversion of b before the Sum is stored in the b-memristor in the third step. After that, we calculate $\overline{ab}$ and store the inversion in the c-memristor. With this, the C_{out} is equal to ab and can be used for the calculation of higher bits. For the SINC algorithm, we require $3n$ steps and $2n + 1$ memristors for an n-bit calculation. The SINC+ algorithm needs $3n + 3$ steps and $2n + 2$ memristors but exhibits a better error-reduction behavior as we will see in later sections.

C. Parallel Topology: PINC/PINC+

The parallel topology consists of n different serial topologies. Each of them can be connected to a shared c-memristor via CMOS switches [19]. In this structure, each bit can be computed in parallel. The only steps that cannot be executed simultaneously are those dependent on the carry-in, as they rely on the outcome of the preceding bit. The exact procedure of PINC and PINC+ is the same as the serial implementations from Table IV and Table V. The operations in blue are again only used in the last bit of the PINC+ variant. But instead of calculating each bit sequentially, we compute them simultaneously. Since the PINC and PINC+ algorithms do not depend on the carry-in, we can completely parallelize each approximated bit. The exact parallel algorithm from [19] first calculates 12 carry-independent steps. So both PINC and PINC+ are executed before the carry-dependent section of the first exact algorithm begins. We visualized this in Figure 3, where we implemented the PINC+ algorithm for the lower bits in an RCA configuration. We can see that the total number of steps only depends on the number of exact adders $n - k$. The same is true for the PINC algorithm. The PINC algorithm only requires 3 steps and $3n$ memristors for an n-bit addition. The PINC+ adder needs 6 steps and $3n + 1$ memristors.

D. Semi-Serial Topology: S-SINC/S-SINC+

The semi-serial topology [25] consists of two parallel rows with the a and b inputs. Both rows can be connected to the carry and work memristors via switches. We will denote the rows with the a-memristors and the b-memristors as Section I and Section II, following the original article [25]. The S-SINC

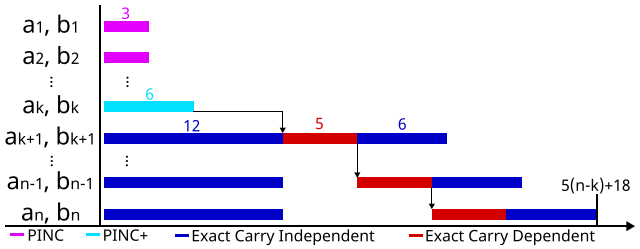


Fig. 3. Number of steps of an RCA in the parallel topology with k PINC+ adder and $n-k$ exact adder from [19].

TABLE VI

EXACT PROCEDURE OF THE S-SINC ALGORITHM IN THE SEMI-SERIAL TOPOLOGY. OPERATIONS COLORED IN BLUE ARE ONLY EXECUTED ONCE

Steps	Section 1	Section 2	Equivalent Logic
-		$w_1 = w_2 = 0$	$False(w_1, w_2)$
switch w_j with w_k $(w_1 \Leftrightarrow w_2)$, $w_j := w_1$, $w_k := w_2$			
1	$w_j^{'} = a \rightarrow w_j{'}$	$w_k = 0$	$w_j = \overline{a}$ & $False(w_k)$
2		$b^{'} = w_j^{'} \rightarrow b$	$b = \overline{a} \rightarrow b = Sum$

TABLE VII

EXACT PROCEDURE OF THE S-SINC+ ALGORITHM IN THE SEMI-SERIAL TOPOLOGY. RED OPERATIONS ARE ONLY CALCULATED ONCE. OPERATIONS COLORED IN BLUE ARE ONLY PERFORMED AT THE LAST APPROXIMATED BIT

Steps	Section 1	Section 2	Equivalent Logic
-		$w_1 = w_2 = 0$	$False(w_1, w_2)$
switch w_j with w_k $(w_1 \Leftrightarrow w_2)$, $w_j := w_1$, $w_k := w_2$			
1	$w_j^{'} = a \rightarrow w_j{'}$	$w_k = 0$	$w_j = \overline{a}$ & $False(w_k)$
-		$w_k^{'} = b \rightarrow w_k$	$w_k = \overline{b}$
2	$w_k^{''} = a \rightarrow w_k^{'}$	$b^{'} = w_j^{'} \rightarrow b$	$b = \overline{a} \rightarrow \underline{b = Sum}$ & $w_k = a \rightarrow \overline{b}$
-	$c^{'} = w_k^{''} \rightarrow c$		$c = a \rightarrow \overline{b} = ab = C_{out}$

algorithm can be seen in Table VI, where we reduced the steps required per bit down to two. This was done with a work memristor switching scheme, where w_1 and w_2 are used for the calculation alternately. The unused work memristor is reset in parallel to this. An additional step before the first bit is required to reset both work memristors.

The S-SINC+ algorithm is shown in Table VII. We expand on the S-SINC approach and calculate the $C_{out} = ab$ with two additional steps. For the last approximated bit, both work memristors are used in parallel. S-SINC requires $2n+1$ steps, and the S-SINC+ algorithm $2n+3$ steps for an n-bit calculation. The S-SINC algorithm uses $2n+2$ memristors and 4 switches. The advanced approach (S-SINC+) needs an additional memristor and two additional switches for the C_{out}.

E. Semi-Parallel Topology: S-PINC/S-PINC+

The semi-parallel topology [24] consists of two parallel rows (sections) with either the input memristors a and b. Each section also contains a work memristor. Additionally, the c-memristor is located in Section II of the circuit. Each section of the circuit can be connected to a resistor via switches (S1 for Section I, and S3 for Section II). With a third switch (S2), the sections can be connected to each other to exchange information. We will refer to this as a calculation "between sections". For more detailed information on the semi-parallel topology please see [24]. The S-PINC algorithm can be seen in Table VIII, which is very similar to SINC. The difference is that the last step is computed between the sections. This

TABLE VIII

EXACT PROCEDURE OF S-PINC IN THE SEMI-PARALLEL TOPOLOGY

Steps	Operation	Section	(S1,S2,S3)	Equivalent Logic
1	$w_1 = 0$	a-Section	(1,0,0)	$False(w_1)$
2	$w_1^{'} = a \rightarrow w_1$	a-Section	(1,0,0)	$w_1 = \overline{a}$
3	$b^{'} = w_1^{'} \rightarrow b$	between	(0,1,1)	$b = \overline{a} \rightarrow b = Sum$

algorithm requires $3n$ steps and $2n+1$ memristors for a n-bit calculation.

In the S-PINC+ algorithm, we use an additional work memristor to save the inversion of b in parallel to the S-PINC steps. After that, we need two steps to calculate ab and save it in the c-memristor. With this, the algorithm needs $2n+3$ memristors and $3n+2$ steps for an n-bit addition. The exact procedure with the switch states can be seen in Table IX.

IV. CIRCUIT-LEVEL SIMULATION

A. Circuit Simulation Setup

To verify the functionality of the algorithms we simulated them at the circuit-level via LT-SPICE. For this, we used a model based on the Voltage-controlled ThrEshold Adaptive Memristor (VTEAM) model [36] implemented in SPICE [25], [48]. In this model, the parameters are set similarly to Table X, which were derived by fitting the model to a real discrete Knowm memristor [49]. This increases our confidence in the practical relevance of our circuit simulations and allows us to compare our work more easily with others who have used the same model. Similar to the differences between discrete and integrated CMOS devices, discrete memristors lead to slower operations and increased power consumption. It is reasonable to expect the integrated memristor devices to have a significantly better operational speed and power efficiency. Due to the lack of access to integrated memristors, we use measurement-fitted models to ensure a realistic and practical implementation of our proposed circuits. We chose the IMPLY specific parameters similar to Table XI to allow for a direct and fair comparison to SoA papers such as [6], [19], [27], and [35].

B. Simulation Results

1) NoCarry: We simulated all NoCarry implementations in their corresponding topology and evaluated all the possible input combinations of the reduced truth table from Table III. We only evaluated these cases, since $C_{out} = 0$ is true for every approximated bit. Our theoretical calculations agree with the simulation results for every implementation we present. Since SINC, PINC, and S-PINC are very similar if we only evaluate one bit, the resulting waveforms are almost identical. They only differ slightly in energy consumption, which is explained in more detail in Section V. The simulation results for these algorithms can be seen in Figure 4. In the first step of every case from $0-30\mu s$ the work memristor w_1 is set to logical '0' via a $False(w_1)$ operation. Then the inversion of a is calculated from $30-60\mu s$, which is saved in w_1. In the last step, the result of the NoCarry algorithm is stored in the b-memristor in the period of $60-90\mu s$. Each approximated bit does not depend on the preceding bits, so the

TABLE IX

EXACT PROCEDURE OF S-PINC+ IN THE SEMI-PARALLEL TOPOLOGY. OPERATIONS COLORED IN BLUE ARE ONLY PERFORMED AT THE LAST APPROXIMATED BIT

Steps	Section 1	Section 2	Between sections	(S1,S2,S3)	Equivalent Logic
1	$w_1 = 0$	$w_2 = 0$	-	(1,0,0) (1,0,1)	$False(w_1)$ & $False(w_2)$
2	$w_1^{'} = a \rightarrow w_1$	$w_2^{'} = b \rightarrow w_2$	-	(1,0,0) (1,0,1)	$w_1 = \overline{a}$ & $w_2 = \overline{b}$
3	-	-	$b^{'} = w_1^{'} \rightarrow b$	(0,1,1)	$b = \overline{a} \rightarrow b = Sum$
-	-	-	$w_2^{''} = a \rightarrow w_2^{'}$	(1,1,0)	$w_2 = \overline{ab}$
-	-	$c^{'} = w_2^{''} \rightarrow c$	-	(0,0,1)	$c = a \rightarrow \overline{b} = ab = C_{out}$

TABLE X

VTEAM SETUP PARAMETER

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

TABLE XI

IMPLY LOGIC PARAMETER

Parameter	V_{SET}	V_{RESET}	V_{COND}	R_G	t_{pulse}
Value	1 V	-1 V	900 mV	40 kΩ	30 μs

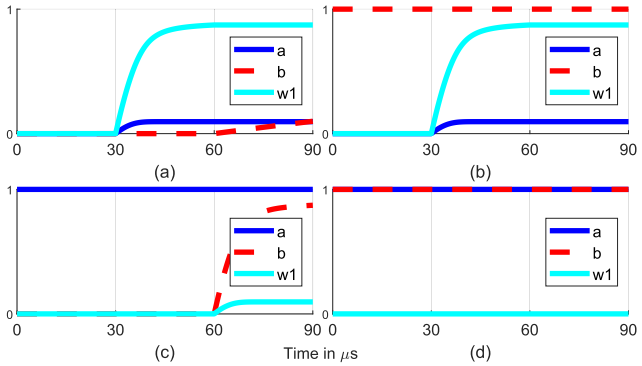


Fig. 4. Simulation of the SINC, SPINC, and PINC adder. *Sum* saved in the b-memristor at 90μs: (a) "ab" = "00", (b) "ab" = "01", (c) "ab" = "10", (d) "ab" = "11" (incorrect case).

c-memristor is unused until the first exact full adder. Since the S-SINC algorithm uses a work memristor switching scheme, as explained in Section III-D, the simulation results differ from other implementations. We choose not to plot this algorithm since most of the cases can be seen in the 4-bit S-SINC+ simulation in Figure 6. The results again agree with our theoretical calculations.

2) NoCarry+: As explained in Section III, the NoCarry+ algorithms differ from NoCarry only in the last approximated bit. In this last bit $Sum = a + b$ and $C_{out} = ab$. Since the previous bits are already discussed in Section IV-B.1 we only discuss the last bit in this section. We validated all possible cases and the results again agree with our theoretical calculations. Since the PINC+ algorithm is a parallelized version of SINC+, the results waveforms are also equal. We assume the c-memristor to be at logical '0' since this algorithm only follows bits that are calculated with the NoCarry algorithms. We plotted a correct case ("ab = 10") and the erroneous case ("ab = 11") in Figure 5. Each step from Table V corresponds to 30μs. Different from SINC, the inversion of b is this time stored in w_2 at $60 - 90\mu s$. After this $Sum = a + b$ is saved in the b-memristor at $90 - 120\mu s$. The resulting $C_{out} = ab$ is then saved in the c-memristor at $150 - 180\mu s$.

We simulated the S-SINC+ algorithm in a 4-bit variant, to better illustrate our memristor switching scheme from Table VII. The memristors are initialized as $w_j := w_1$ and $w_k := w_2$. The resulting waveforms are shown in Figure 6. From $0 - 30\mu s$ both work memristors are reset to logical '0'. After that three bits with different combinations are calculated, where each bit needs 60μs. The Sum_i results are stored in the b_i-memristor at 90, 150, and 180μs. The fourth bit requires four steps since the S-SINC+ algorithm from Table VII is applied. As the input combination is "ab = 11", the incorrect (by design) $Sum = 1$ is stored between $270 - 300\mu s$. The correct $C_{out} = 1$ is saved between $300 - 330\mu s$ in the c memristor.

The results of S-PINC+ are shown in Figure 7, where the correct case "ab = 00" and the erroneous case "ab = 11" are presented. The *Sum* result is again stored in the b-memristor at $60 - 90\mu s$. In comparison to the S-PINC algorithm, two additional steps are required to calculate the C_{out}, which is saved in the c-memristor at $90 - 150\mu s$.

C. Further Experiments

We conducted additional experiments to validate the proposed full adders' functionality at circuit-level. We embedded each algorithm as the lower four bits in an 8-bit RCA configuration. The corresponding exact algorithm of the topology implements the higher bits. We evaluated all possible input combinations. The results agree with our theoretical calculations. Since with our approach, each approximated bit is independent from its predecessor this was expected. Our NoCarry+ implementations propagate a carry-in to the exact adders, which also does not lead to any false results. Since memristors are non-ideal devices a deviation in R_{on} and R_{off} from their nominal values can be expected in practice. To encompass this non-ideality in our experiments, we repeated our simulations with deviations from ±5% to ±20%. The results align with previous experiments, indicating that our algorithms are functional despite such practical non-idealities. Since we validated the functionality of the fundamental building blocks (adders) at circuit-level, for more complex system-level evaluations that use these building blocks, we will use a behavioural-level model in the rest of this paper.

V. ERROR ANALYSIS

Error metrics such as MED, NMED, and MRED are used to evaluate and compare the erroneous behavior of approximated adders [6], [35]. The exact definition can be seen in

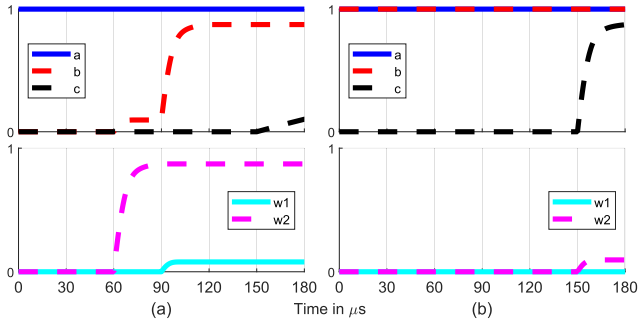


Fig. 5. Simulation results of SINC+/PINC+ for two input combinations (a) "ab = 10", (b) "ab = 11" (*Sum* incorrect).

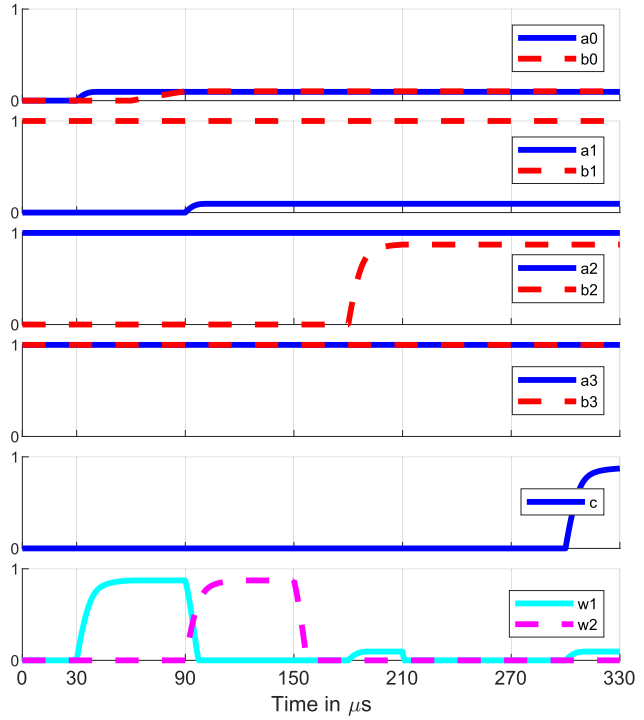


Fig. 6. 4-bit simulation of S-SINC+, with a_{3-0} = "0011" and b_{3-0} = "0101". Each step takes 30μs. *Sum* results are stored in the b-memristors at 90, 150, 210, and 330μs. C_{out} is only calculated for the last bit between 270-330μs.

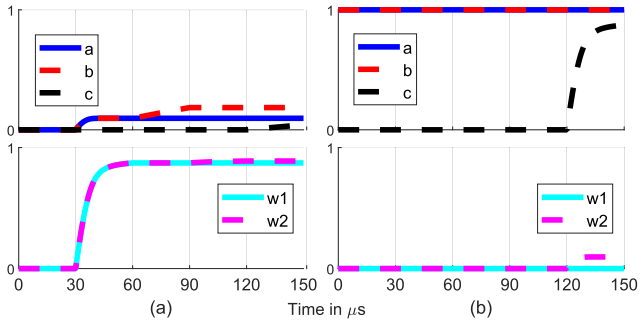


Fig. 7. Simulation results of S-PINC+ for two input combinations (a) "ab = 00", (b) "ab = 11" (*Sum* incorrect).

Equation (5) - Equation (7), with detailed information in [10], [13], and [38].

$$MED = \frac{1}{2^{2n}} \cdot \sum_{i=1}^{2^{2n}} |SUM_{Exact} - SUM_{Ax}|_i \tag{5}$$

$$NMED = \frac{MED}{2^{n+1} - 1} \tag{6}$$

$$MRED = \frac{1}{2^{2n}} \cdot \sum_{i=1}^{2^{2n}} \frac{|SUM_{Exact} - SUM_{Ax}|_i}{SUM_{Exact,i}} \tag{7}$$

We created a behavioural-level MATLAB model, where we embed the approximated adders as lower bits in an RCA. The approximation degree k/n denotes that k adders based on our algorithms are implemented as the lower bits out of a total of n bits. We evaluated the error metrics of these partially approximated RCA for 8/16/32-bit at varying approximation degrees, with Cin = '0'. The results of the MED, NMED, and MRED for all the aforementioned combinations are shown in Table XII.

A. Error Metrics for 8-Bit RCA

For the 8-bit case, we applied all 65536 different input combinations to partially approximated RCA. We repeated this procedure with different approximation degrees. We observe that the MED and thus also the NMED is roughly doubled per additional approximated bit. The NoCarry+ approximation exhibits better NMED and MRED than the NoCarry variant for all approximation degrees over 1/8. The difference between these two approaches is increasing exponentially for all metrics evaluated. This implies that the NoCarry+ adder produces progressively lower relative errors as the approximation degree increases. With an approximation degree of 8/8 both approximations exhibit very good error-containing behavior. They display a MRED of roughly 0.21 for the NoCarry and 0.17 for the NoCarry+ adders. This indicates that the error relative to the exact results averages 21% and 17% of the exact values.

B. Error Metrics for 16/32-Bit RCA

To analyze the 16-bit and 32-bit RCA, we generated one million random combinations for the inputs. We did this since a complete evaluation would require 2^{2n} combinations, which is computationally expensive. So the results for the 16/32-bit error metrics may underlie a stochastic deviation. We again evaluated a partially approximated RCA with varying approximation degrees. We presented the results of the error metrics in Table XII in such a way that they include the same percentage of approximated adders in each row. With this, we can see that with fewer approximated adders the 16/32-bit systems yield much lower error metrics. The difference increases with a rising approximation degree but diminishes when only approximated adders are used. This indicates that a full adder with a higher number of bits yields much lower errors.

VI. Circuit-Level Comparison

We compare the algorithms proposed in this work with the exact full adders from [19], [24], [25], and [27] and the approximated adders from [6], [34], and [35]. We evaluated various relevant circuit-level metrics such as speed, energy consumption, and area usage. We define an improvement of the parameter P in the rest of this work by Equation (8).

$$Improvement = \frac{P_{worse} - P_{better}}{P_{worse}} \times 100\% \tag{8}$$

TABLE XII
ERROR METRICS OF THE PROPOSED ALGORITHMS EMBEDDED IN A8/16/32-BIT RCA WITH DIFFERENT APPROXIMATION DEGREES

Approximated Full Adder	**8-bit RCA**			**16-bit RCA**			**32-bit RCA**		
	MED	NMED	MRED	MED	NMED	MRED	MED	NMED	MRED
	Approximation degree 1/8			Approximation degree 2/16			Approximation degree 4/32		
No Carry	0.25	0.00049	0.0013	0.7879	0.0000060	0.000017	3.8829	<e-09	<e-08
No Carry +	0.25	0.00049	0.0013	0.6083	0.0000046	0.000013	2.9694	<e-09	<e-09
	Approximation degree 2/8			Approximation degree 4/16			Approximation degree 8/32		
No Carry	0.75	0.0015	0.0040	3.6904	0.000028	0.000078	65.5187	<e-08	<e-07
No Carry +	0.625	0.0012	0.0034	2.9103	0.000022	0.000065	48.7034	<e-08	<e-07
	Approximation degree 3/8			Approximation degree 6/16			Approximation degree 12/32		
No Carry	1.75	0.0034	0.0092	15.7382	0.00012	0.00033	1011	<e-06	<e-06
No Carry +	1.375	0.0027	0.0073	12.1559	0.000093	0.00025	753.45	<e-07	<e-06
	Approximation degree 4/8			Approximation degree 8/16			Approximation degree 16/32		
No Carry	3.75	0.0073	0.0191	65.9018	0.00050	0.0014	1643	0.0000019	0.0000052
No Carry +	2.875	0.0056	0.0149	47.5004	0.00036	0.0010	1215	0.0000014	0.0000038
	Approximation degree 5/8			Approximation degree 10/16			Approximation degree 20/32		
No Carry	7.75	0.0152	0.0377	261.2083	0.0020	0.0054	2.652e+05	0.000031	0.000081
No Carry +	5.875	0.0115	0.0293	187.0573	0.0014	0.0040	2.047e+05	0.000024	0.000065
	Approximation degree 8/8			Approximation degree 16/16			Approximation degree 32/32		
No Carry	63.75	0.1248	0.2116	15677	0.1196	0.2056	1.110e+09	0.1298	0.2179
No Carry +	47.875	0.0937	0.1739	12232	0.0933	0.1737	7.958e+08	0.0926	0.1727

A. Comparison to Exact Adders

1) Energy Consumption: We used the LT-SPICE energy consumption tool for all proposed algorithms. Our simulation encompassed all feasible input combinations of a full adder. We present the mean values over all simulations. The last bit of the NoCarry+ adders in the different topologies were calculated separately. The first step of the S-SINC/S-SINC+ algorithms was also evaluated on its own. The result of 0.2591nJ was defined as the average of the possible input state combinations of the work memristors. We recreated the exact algorithms from [24], [25], [27], and [19] with specific simulation values we used in this work to have a fair comparison. The formulas for the NoCarry implementations in an RCA structure, with k approximated and n total adders, are illustrated in Equation (9) to Equation (12), measured in nJ.

$$E_{SINC}(n,k) = 0.7230k + 4.8250(n-k) \tag{9}$$
$$E_{PINC}(n,k) = 0.7230k + 4.0772(n-k) \tag{10}$$
$$E_{SSINC}(n,k) = 0.5714k + 3.8435(n-k) + 1.0617 \tag{11}$$
$$E_{SPINC}(n,k) = 0.6372k + 4.8339(n-k) \tag{12}$$

Since the NoCarry+ implementations differ only for the last approximated bit, the energy consumption remains the same except for the increase in energy usage for this bit. The energy consumption of the SINC+, PINC+, S-SINC+, and S-PINC+ algorithms is increased by 0.7844nJ, 0.7844nJ, 0.8024nJ, and 0.9287nJ respectively. The proposed approach saves up to 54% of energy when compared to the corresponding exact algorithm. The improvements based on Equation (8), to the corresponding exact algorithms are shown in Table XIV. The overall most energy-efficient adder is SINC, followed by PINC.

2) Number of Steps: The second important metric at circuit-level is the number of steps (clock cycles) required per bit. These represent the delay of the calculation. The required steps of the NoCarry and NoCarry+ implementations, with k approximated out of n total adders, are shown in Equation (13) to Equation (16). The additional steps of the NoCarry+ implementations are marked in blue.

$$S_{SINC} = 3k + 22(n-k)\ (+3) \tag{13}$$
$$S_{PINC} = 5(n-k) + 18 \tag{14}$$
$$S_{SSINC} = 2k + 10(n-k) + 3\ (+2) \tag{15}$$
$$S_{SPINC} = 3k + 17(n-k)\ (+2) \tag{16}$$

With the approach from this paper up to 54% of the steps are savable with an approximation degree of 5/8. PINC and PINC+ are the fastest and only require a mere 33 steps for an 8-bit addition. It is worth noting that the parallel algorithms from this work are the only implementations that require the same number of steps. This is attributed to the property of the parallel topology, where only a small portion of the steps need to be computed sequentially. We illustrated this in Figure 3. Any approximation with $k = 5$ would achieve the same results if the longest carry-chain of the lower bits were under 12 steps. The SINC+ and S-PINC+ require three more steps than the associated NoCarry implementation. The S-SINC+ only needs two additional cycles. All of our proposed implementations reduce the number of steps by at least 43% compared to the corresponding exact algorithms. The speed improvements for all implementations can be seen in Table XIV.

3) Area Usage: Another comparison point at circuit-level is the area required per bit. This is evaluated by comparing the number of memristors and CMOS switches. In the serial, semi-serial, and semi-parallel topology, the work resistors are reused. So they depend on the exact algorithms. This means that no improvements can be achieved here. In the parallel adder structure, each row is calculated separately. Thus, we can reduce the number of memristors in each approximated bit. The PINC algorithm saves 1 memristor per approximated bit. PINC+ also saves 1 memristor per bit, besides the last approximated one. The number of switches can also only be reduced in the parallel topology. Each approximated bit in PINC does not require a switch since they do not calculate the carry-out. This is explained in more detail in Section III-C. The most area-efficient are all algorithms in the serial topology since they only require $2n + 3$ memristors

TABLE XIII
CIRCUIT-LEVEL COMPARISON TO SOA FULL ADDER

Algorithm	Energy consumption (nJ)		No. of steps		No. of memristors		No. of switches	
	n, k	n=8-bit, k=5	n, k	n=8-bit, k=5	n, k	n=8-bit, k=5	n, k	n=8-bit, k=5
Serial Exact 1 [27]*	4.8250n	38.6000	22n	176	2n+3	19	0	0
Serial Exact 2 [19]*	4.0772n	32.6176	23n	184	2n+3	19	0	0
Parallel Exact [19]*	4.0772n	32.6176	5n + 18	58	4n+1	33	n	8
Semi-Serial Exact [25]	3.8435n + 0.8053	31.5580	10n + 2	82	2n+6	22	12	12
Semi-Parallel Exact [24]*	4.8339n	38.6712	17n	136	2n+3	19	3	3
SIAFA 1,3 [35]*	1.7090k + 4.8250(n-k)	23.0200	8k + 22(n-k)	106	2n+3	19	0	0
SIAFA 2 [35]*	2.5131k + 4.8250(n-k)	27.0405	10k + 22(n-k)	116	2n+3	19	0	0
SIAFA 4 [35]*	1.7066k + 4.8250(n-k)	23.0080	8k + 22(n-k)	106	2n+3	19	0	0
SAFAN [34]*	1.6628k + 4.8250(n-k)	22.7890	7k + 22(n-k)	101	2n+3	19	0	0
Seiler [6]	1.6678k + 3.8435(n-k) + 0.865	20.7345	5k + 10(n-k) + 3	58	2n+6	22	12	12
SINC	0.7230k + 4.8250(n-k)	18.0900	3k + 22(n-k)	81	2n+3	19	0	0
SINC+	0.7230k + 4.8250(n-k) + 0.7844	18.8744	3k + 22(n-k) + 3	84	2n+3	19	0	0
PINC	0.723k + 4.0772(n-k)	15.8466	5(n-k) + 18	33	3k+4(n-k)+1	29	n-k	3
PINC+	0.723k + 4.0772(n-k) + 0.7844	16.6310	5(n-k) + 18	33	3k+4(n-k)+2	30	n-k+1	4
S-SINC	0.5714k + 3.8435(n-k) + 1.0691	15.4566	2k + 10(n-k) + 3	43	2n+6	22	12	12
S-SINC+	0.5714k + 3.8435(n-k) + 1.8715	16.2590	2k + 10(n-k) + 5	45	2n+6	22	12	12
S-PINC	0.6372k + 4.8339(n-k)	17.6877	3k + 17(n-k)	66	2n+3	19	3	3
S-PINC+	0.6370k + 4.8339(n-k) + 0.9287	18.6164	3k + 17(n-k) + 2	68	2n+3	19	3	3

* We have simulated these adders with the specified parameters from Section IV-A to allow for a fair comparison.

and no switches. An overview of the improvements in comparison to the corresponding exact algorithms can be seen in Table XIV.

B. Comparison to Approximated Adders

In this section, we only compare our work to the SoA IMPLY-based approximated adders from [6], [34], [35]. We do this, since the disparity to other logic forms, such as MRL approximations [45], [46], is too significant to make a fair and meaningful comparison. A complete overview of different circuit-level comparison points can be seen in Table XIII. All of the following comparisons are made with an approximation degree of 5/8.

1) Energy Consumption: The SINC and SINC+ from this work require less energy than any SoA approximated adders in the serial topology. Our approaches require $18\% - 33\%$ less energy than the SIAFA adders from [35] and $17\% - 21\%$ less than the SAFAN adder from [34]. In the semi-serial topology, our S-SINC and S-SINC+ implementations are 22% and 26% more efficient than the adder from [6]. To our knowledge, we presented the only approximated adders in the parallel and semi-parallel topology. So the comparison is not 100% fully fair, due to them being implemented in a different topology. Compared to SoA adders our semi-parallel and parallel approaches are $10\% - 41\%$ more energy efficient.

2) Number of Steps: Compared to the adders from [34] and [35], our algorithms in the serial topology require $17\% - 30\%$ less steps. The semi-serial adders from this work improve the speed to [6] by $22\% - 26\%$. If we compare all of the proposed adders to the SoA an improvement of up to 72% is reached. Our serial implementations are $28\% - 31\%$ slower than the semi-serial adders from [6]. The reason is that the serial topology is intrinsically 53% slower than the semi-serial topology [25]. We plotted the energy consumption and the number of steps of the SoA and the proposed algorithms in Figure 8(c). We grouped the approximations corresponding to their exact algorithms. We can see that the NoCarry and NoCarry+ implementations are at the Pareto front and drastically improve over the SoA.

TABLE XIV
CIRCUIT-LEVEL IMPROVEMENTS TO EXACT SOA ADDERS WITH $n = 8, k = [1, 5]$

Algorithm	Compared to	Improvement to exact adder [%]			
		Energy consumption	No. of steps	No. of memristors	No. of switches
SINC	Serial 1 [27]	11–53	11–54	0	0
SINC+	Serial 1 [27]	9–51	9–52	0	0
PINC	Parallel [19]	10–51	9–43	3–12	13–63
PINC+	Parallel [19]	8–49	9–43	0–9	0–50
S-SINC	Semi-Serial [25]	10–51	9–48	0	0
S-SINC+	Semi-Serial [25]	7–49	6–45	0	0
S-PINC	Semi-Parallel [24]	11–54	10–52	0	0
S-PINC+	Semi-Parallel [24]	9–52	9–50	0	0

3) Area Usage: All the approximated algorithms are embedded in an RCA as the lower bits. So the number of memristors and switches depends on the exact algorithm used for the higher bits. Since for the serial and semi-serial topology, we used the same exact algorithms as the SoA, the area usage is also equal. From an area consumption point of view, our approach cannot improve the SoA. The proposed parallel implementations require $24\% - 37\%$ more memristors than SoA approximations. Compared to algorithms in the serial topology, our parallel implementations also require additional switches.

4) Error Metrics: We compared our approaches to the SoA via the error metrics MED, NMED, and MRED. Our algorithms outperform the SoA at every approximation degree. We plotted the NMED and MRED for an 8-bit adder with varying approximation degrees in Figure 8(a) and Figure 8(b). It is noticeable that at $k \leq 4$ the different approximations show very similar results. At $k = 5$, NoCarry and NoCarry+ exhibit a NMED lower than SoA by $12\% - 56\%$ and a MRED lower by $27\% - 64\%$. This difference in NMED and MRED rises exponentially to $25\% - 76\%$ and $64\% - 81\%$ when all adders are approximated. This indicates that the proposed adders show a much better error-reduction behavior than the SoA approximations.

VII. APPLICATION IN IMAGE PROCESSING

One of the most prevalent applications of AxC is image processing. It has an inherent resilience to errors

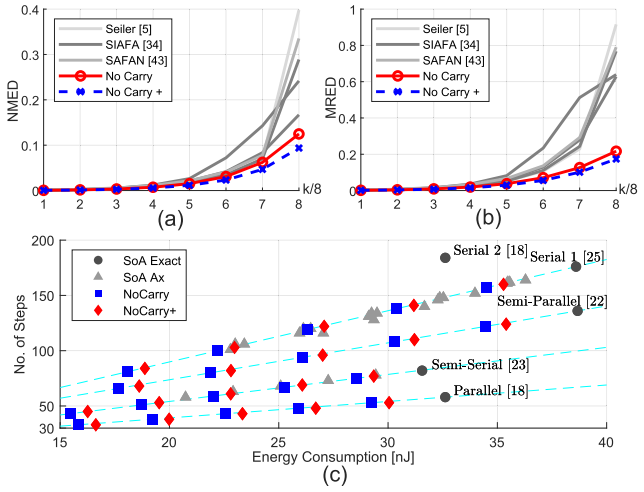


Fig. 8. Comparison to SoA Ax adders: (a) NMED, (b) MRED, (c) Energy consumption to No. of steps with $k/n = 5/8$.

TABLE XV
AVERAGE QUALITY METRICS OF IMAGE PROCESSING APPLICATIONS

Ax Algorithm	Image addition		Grayscale filter		Image subtraction	
	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM
Approximation degree 1/8						
No Carry	Inf	1	Inf	1	49.08	0.9752
No Carry +	54.04	0.9989	47.97	0.9967	52.40	0.9898
Approximation degree 2/8						
No Carry	54.31	0.9990	51.70	0.9981	46.16	0.9596
No Carry +	54.31	0.9990	47.04	0.9961	49.74	0.9886
Approximation degree 3/8						
No Carry	46.34	0.9951	44.38	0.9926	42.26	0.9193
No Carry +	48.04	0.9952	42.75	0.9895	44.85	0.9715
Approximation degree 4/8						
No Carry	39.71	0.9825	37.39	0.9722	38.16	0.8615
No Carry +	42.10	0.9826	37.67	0.9690	39.41	0.9108
Approximation degree 5/8						
No Carry	33.90	0.9521	30.80	0.9169	34.15	0.8053
No Carry +	36.39	0.9512	32.34	0.9162	35.16	0.8303
Approximation degree 6/8						
No Carry	27.84	0.8849	24.71	0.8168	29.16	0.7364
No Carry +	30.50	0.8856	26.53	0.8157	30.51	0.7635

and versatile applicability across various domains such as medicine, robotics, and automation [44]. Given the inherent computational complexity of these applications, adopting an approximated approach holds promise for significant improvements in speed and energy efficiency. These are important as they are critical considerations in this domain. We simulated the proposed approaches on the behavioral level with varying approximation degrees. We evaluated common applications such as image addition and grayscale filtering. For both of these, we introduce datasets with common example images [50]. We further optimized our algorithms for image subtraction and evaluated them over the background subtraction dataset from [51]. To validate our approaches on more complex tasks, we embed our approximated adders in multipliers and use them for Gaussian smoothing. We used the common quality metrics PSNR and MSSIM. In the literature, a PSNR threshold of 30dB is deemed acceptable for the human visual system [37]. The average quality results over the corresponding dataset of all simulated applications are presented in Table XV.

A. Image Addition

One of the fundamental image processing applications is the addition of images, often used in tasks such as masking or enhancement [14]. We used the RCA with varying approximation degrees to add every pixel of two 8-bit grayscale images and half the result. For the image addition dataset [50], we combined 21 common images with varying content from humans to landscapes and resized them to 256×256 pixels. We calculated 100 additions with random pairings and noted the average quality results with the corresponding deviation. We can see in Table XV that with up to 5/8 approximated adders, both proposed approaches reach the 30dB threshold of PSNR. With six adders approximated the NoCarry+ algorithm reaches the PSNR threshold on average. However, with a standard deviation of 1.21 dB, it remains impractical for many image combinations. It is interesting to notice that with only one NoCarry adder the addition is calculated with full accuracy. This is attributed to the reduced error propagation of incorrect cases and the rounding errors in addition. We can also observe that the deviation of the quality metrics rises with a higher approximation degree. The results of an example addition with 5/8 approximated adders are shown in Figure 9 (a-c).

B. Gray-Scale Filter

The grayscale filter transforms a colored RGB image into a grayscale by adding the three color spaces pixel-wise, via two additions. Common applications of the gray-scale filter include medical imaging and computer vision. Since in common grayscale conversion methods, the colors are not equally weighted we first added the red and blue pixels and halved this result. We then added the green color space to the result, to better represent that green is a more relevant color to the human visual system. We evaluated the SoA and the proposed adders over our gray-scale filtering dataset. We included 21 common images with varying content from animals to groceries and different sizes. A more detailed explanation can be found in [50]. We can see in Table XV that the gray-scale filter yields worse results than image addition. Since our algorithms are OR-based, adding three numbers leads to a higher chance of the erroneous case occurring. This then leads to worse results. With up to 5/8 approximated adders this application reaches the 30dB PSNR threshold. We represented this by an example in Figure 9.

C. Image Subtraction

Image subtraction is commonly used for surveillance and motion detection, subtracting images in a sequence. The subtraction procedure is normally implemented by inverting the subtrahend bitwise and then adding it to the minuend. The carry-in is set to logical '1'. Since the proposed adders in this work do not depend on the carry-in, it is set to '0'. For this application, we can further optimize our NoCarry approximation to save additional steps and energy. Instead of inverting the bits of the minuend, we use the properties of IMPLY logic to simplify the approximated subtraction.

Our approximation is based on a_i OR b_i, which is equivalent to $\overline{a_i} \rightarrow b_i$ in IMPLY logic. When we now not invert a_i

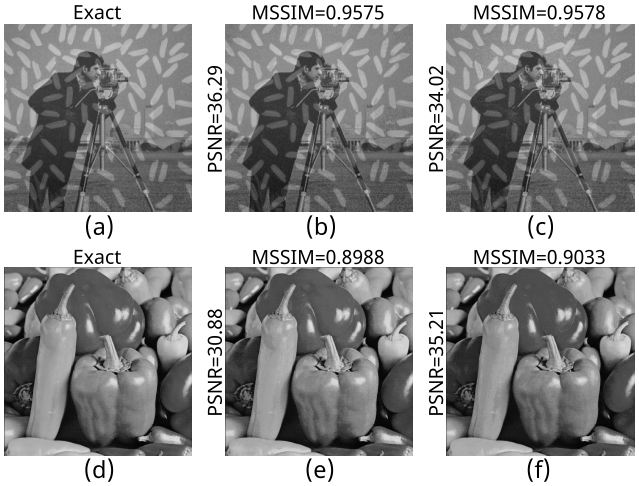


Fig. 9. Example Results: Exact, NoCarry 5/8 Ax FA, NoCarry+ 5/8 Ax FA for (a-c) image addition of rice + cameraman [52], (d-f) grayscale filter of peppers [52].

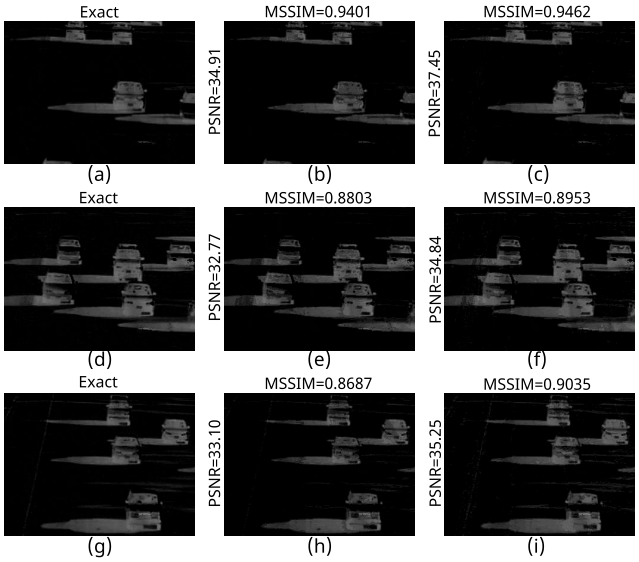


Fig. 10. Image subtraction examples of "highway" dataset [51] with exact, NoCarry 5/8 Ax FA, NoCarry+ 5/8 Ax FA: (a-c) image 1, (d-f), image 143, (g-i) image 420.

beforehand we can simplify the approximated bits of the subtraction to $Sub_i = b_i - a_i \approx a_i \rightarrow b_i$. With this approach, we reduce the required memristors from $3n$ (depending on the topology) to only $2n$ for n approximated bits. The number of steps for SINC and S-PINC is also reduced from $3n$ down to n. Considering the saved steps for not inverting the minuend, the savings would be increased by an additional step for each bit. This method is applicable to the serial, parallel, and semi-parallel topology. This optimization does not work for the semi-serial implementation since the a and b memristors can not be directly connected.

We simulated this application on the background subtraction dataset, called "highway", from [51]. It includes 439 images which we converted to grayscale beforehand. We then subtracted the background image from each entry with image subtraction. We see in Table XV that with approximation degrees of up to 5/8, both approaches meet the 30dB PSNR requirement. The NoCarry+ adder surpasses the threshold with six approximated adders. But with a deviation of over 1dB, this still does not lead to acceptable results for many images. We showed three example results of our experiments in Figure 10. We can see that they are rather indistinguishable from the exact results.

TABLE XVI
AVERAGE IMAGE QUALITY COMPARISON TO SOA WITH 5/8 APPROXIMATED ADDERS OVER DIFFERENT DATASETS

Algorithm	Addition		Grayscale		Subtraction	
	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM
Seiler [6]	32.96	0.8952	31.66	0.8727	30.59	0.6387
SIAFA 1 [35]	32.80	0.8932	30.91	0.8727	31.07	0.7659
SIAFA 2 [35]	28.46	0.8179	27.67	0.8361	24.60	0.4496
SIAFA 3 [35]	32.67	0.8896	31.01	0.8224	30.97	0.7516
SIAFA 4 [35]	31.90	0.8927	29.91	0.8694	30.90	0.7262
SAFAN [34]	30.81	0.9282	25.55	0.8778	30.06	0.6366
No Carry	33.90	**0.9521**	30.80	**0.9169**	34.15	0.8053
No Carry+	**36.39**	0.9512	**32.34**	0.9162	**35.16**	**0.8303**

D. Application-Level Comparison

1) Comparison With Exact Adders: For image addition and the gray-scale filter, acceptable results were reached with approximation degrees of up to 5/8. We will now look at the gains at circuit-level. The proposed adders can save $6\% - 54\%$ of steps and $7\% - 54\%$ of energy depending on the approximation degree and topology. This was calculated by summing up the gains for each pixel. For the addition of two 256×256.8-bit images in Figure 9, our approach saves up to 1.4mJ of energy and 6 million steps. For the 512×512 grayscale conversion example, we were able to save up to 11mJ and 50 million steps. This is significantly higher because this operation consists of two addition operations and the image is four times the size.

In image subtraction, our NoCarry algorithm can be further optimized so that it only requires one step per approximated bit. We simulated the average energy consumption of $a \rightarrow b$ for the serial, parallel, and semi-parallel topologies. The optimized approximated serial and parallel algorithms require 0.4618nJ and the semi-parallel 0.4609nJ. When embedded in an RCA with 5/8 approximated adders, the total energy is 16.7840nJ in the serial, 14.5406nJ in the parallel, and 16.8062nJ in the semi-parallel topology. The number of steps for this configuration is 71 for the serial and 56 for the semi-parallel adder. The number of required cycles for the parallel topology is unchanged due to the dependency on the exact bits. We explained this in more detail in Section III-C. The RCA with 5/8 optimized approximated adders need up to 57% less energy and up to 60% fewer steps compared to the exact algorithms. With our optimization an additional performance improvement for our already efficient approach is possible. For the examples from the "highway" dataset [51] our serial approach achieves the highest gains. The energy consumption is reduced from 3mJ to 1.3mJ and the number of steps from 13.5 to 5.5 million, which is a substantial performance improvement. This is especially useful for resource-constrained applications.

2) Comparison With Approximated Adders: We compare to the IMPLY-based approximations from [6], [34], and [35]. We evaluated with 5/8 approximated adders since this setup still leads to acceptable image qualities. For image addition and grayscale filtering our approaches are $9\% - 43\%$ more energy efficient and require up to 72% fewer steps. This gain is

explained in more detail in Section VI-B. In image subtraction, our approximation is implemented even more efficiently. Our approach can improve energy efficiency by $19\% - 46\%$ when compared to other approximations.

We evaluated the SoA adders over the proposed datasets for addition and grayscaling and the "highway" dataset from [51] to achieve a fair comparison. In Table XVI, we presented the average results for each approach with 5/8 approximated adders. For the image addition dataset, our NoCarry and NoCarry+ approaches surpass the SoA solutions. Our approaches improve over the best PSNR result of [6] by 1dB and 3.4dB, which is a significant increase. Our adders also outperform the SoA in terms of MSSIM in which the NoCarry approach yields the best results. In the grayscaling application our NoCarry+ approach exhibits the best PSNR results with 32.34dB, whereas NoCarry yields the best MSSIM. The PSNR of NoCarry does not improve over SoA adders in this case. The most significant improvement was achieved in image subtraction. Our approximations yield a higher average PSNR by at least 3dB and 4dB for NoCarry and NoCarry+ while improving the MSSIM too.

E. Gaussian Smoothing

The previously covered image-processing applications can executed with just one adder structure. With this case study, we intend to validate the applicability of our partially approximated RCA on a more complex application. We embedded our approximated adders with varying approximation degrees in an unsigned array multiplier, which is explained in [10] and [38]. When separating the rows of an unsigned 8×8 array multiplier, seven 8-bit additions are required to sum them up. These additions can be implemented with an approximated RCA. We will denote the approximation degree with the 7-tuple $\mathcal{K} = (k_7, \ldots, k_1)$, representing the approximation degree $k_i/8$ for the i-th row starting from the top. The i-th row consists of the logical AND connection between the binary input vector a_{7-0} and b_i. We choose to implement Gaussian smoothing since it is often used for image denoising and as a pre-processing stage for various machine learning and computer vision tasks. It is a common procedure that can tolerate errors too [53]. We used a two-dimensional rotationally symmetric 3×3 Gaussian low-pass filter with a standard deviation of 1.5 that we convolve with the input image. We follow the example from [53] and [54] and choose our kernel as shown in Table XVII, since it is considered appropriate for an 8-bit multiplier [54].

We blurred the test image "boat" using the proposed array multiplier with varying $\mathcal{K}$. We evaluated our algorithms and SoA approximations from [6], [34], and [35]. The image quality for some example configurations is presented in Table XVIII. When the whole row is approximated both approaches from this paper have equal results. Since for the advanced approach $C_{out} = ab$ and $b = 0$ for the highest bit per row, no carry-out is ever produced. Our NoCarry and NoCarry+ approaches are superior to the SoA in both PSNR and MSSIM. With five out of seven additions completely approximated ($\mathcal{K}_5$) both of our approaches outperform the SoA by at least 10dB in PSNR. Even with five additions fully and the other additions half approximated ($\mathcal{K}_{544}$), our advanced approach reaches 32.85dB. This indicates that even when most of the full adder units in the multiplier use our NoCarry+ approach the result is still acceptable. We illustrated the results of our approaches with different $\mathcal{K}$ in Figure 11. Our results indicate, that up to 86% of an array multiplier can be approximated, while still achieving acceptable results. This example aims to show the applicability of the proposed algorithms to more complex tasks.

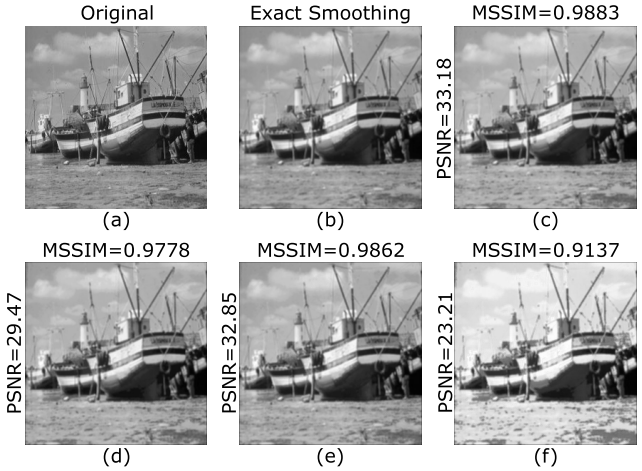


Fig. 11. Gaussian smoothing of "boat": (a) Original image, (b) Exact multiplier, (c) Both ($\mathcal{K}_5$), (d) No Carry ($\mathcal{K}_{544}$), (e) No Carry+ ($\mathcal{K}_{544}$), (f) Both ($\mathcal{K}_6$).

TABLE XVII
GAUSSIAN SMOOTHING 3×3 KERNEL [53], [54]

97	121	97
121	151	121
97	121	97

TABLE XVIII
IMAGE QUALITY OF GAUSSIAN SMOOTHING WITH AN APPROXIMATED ARRAY MULTIPLIER

Ax Algorithm	$\mathcal{K}$	PSNR (dB)	MSSIM
Proposed			
Both	$\mathcal{K}_1$=(8,0,0,0,0,0,0)	64.22	0.9999
Both	$\mathcal{K}_2$=(8,8,0,0,0,0,0)	57.85	0.9995
Both	$\mathcal{K}_3$=(8,8,8,0,0,0,0)	52.57	0.9987
Both	$\mathcal{K}_4$=(8,8,8,8,0,0,0)	42.20	0.9976
Both	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	**33.18**	**0.9883**
Both	$\mathcal{K}_6$=(8,8,8,8,8,8,0)	23.21	0.9137
No Carry	$\mathcal{K}_{444}$=(8,8,8,8,4,4,0)	33.61	0.9882
No Carry+	$\mathcal{K}_{444}$=(8,8,8,8,4,4,0)	39.29	0.9946
No Carry	$\mathcal{K}_{54}$=(8,8,8,8,8,4,0)	30.02	0.9795
No Carry+	$\mathcal{K}_{54}$=(8,8,8,8,8,4,0)	33.16	0.9864
No Carry	$\mathcal{K}_{544}$=(8,8,8,8,8,4,4)	29.47	0.9778
No Carry+	$\mathcal{K}_{544}$=(8,8,8,8,8,4,4)	32.85	0.9862
SoA			
SIAFA1 [35]	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	21.59	0.8883
SIAFA2 [35]	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	17.85	0.6517
SIAFA3 [35]	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	23.14	0.8977
SIAFA4 [35]	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	18.97	0.8960
SAFAN [34]	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	14.88	0.8523
Seiler [6]	$\mathcal{K}_5$=(8,8,8,8,8,0,0)	19.39	0.8923

VIII. CONCLUSION

In this work, we presented two approximated addition concepts for implementation using memristive IMPLY logic and mapped them to a total of eight algorithms on the serial, parallel, semi-serial, and semi-parallel topology. We applied

them in in-memory image processing and showed their benefits. We utilized the unique advantages of each adder structure to reduce the number of steps, energy, and area consumption. Our approach reduces $8\% - 54\%$ of the energy consumption, $6\%-54\%$ of the steps, and up to 12% of memristors, compared to the exact adders. Compared to other approximated adders, our approach is $9\%-43\%$ more energy efficient and improves the speed by up to 72%. We achieved this while yielding a higher accuracy, in terms of both NMED and MRED. To our knowledge, we present the first approximated adders in the semi-parallel and parallel topology, with which were able to decrease the number of steps of an 8-bit addition to only 33.

We embedded our adders as the lower bits of an RCA, verified their functionality, and assessed the error metrics. We applied these partially approximated RCAs in different real-world image processing applications. For image addition and grayscale filtering, we proposed new datasets to diversify our experiments. We achieved acceptable PSNR results of over 30dB with up to 5/8 approximated adders. We improved the energy consumption by $9\% - 43\%$, the number of steps by up to 72%, and the image quality by up to 3.4dB of PSNR when compared to SoA approximations. In the image subtraction application, we optimized our NoCarry algorithm so that it only requires one cycle per approximated bit. With this, we increased the energy efficiency by up to 57%. Our approach also yields significantly improved image quality over SoA approximations by $3\text{dB}-10\text{dB}$ of PSNR. We integrated our approximated adders in an array multiplier and evaluated them on Gaussian smoothing, as an example of more complex end applications. Our results indicate that up to 86% can be approximated with the proposed method, while still achieving acceptable results. An in-depth analysis of more application-specific approximations and their design are domains of future research.

ACKNOWLEDGMENT

The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

REFERENCES

[1] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 409–414.

[2] W. Liu, F. Lombardi, and M. Schulte, "A retrospective and prospective view of approximate computing," *Proc. IEEE*, vol. 108, no. 3, pp. 394–399, Mar. 2020.

[3] R. H. Dennard, J. Cai, and A. Kumar, "A perspective on today's scaling challenges and possible future directions," *Solid-State Electron.*, vol. 51, no. 4, pp. 518–525, 2007.

[4] N. TaheriNejad, "In-memory computing: Global energy consumption, carbon footprint, technology, and products status quo," in *Proc. 24th IEEE Int. Conf. Nanotechnol. (IEEE-NANO)*, 2024, pp. 1–6.

[5] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[6] F. Seiler and N. TaheriNejad, "An IMPLY-based semi-serial approximate in-memristor adder," in *Proc. IEEE Nordic Circuits Syst. Conf. (NorCAS)*, Oct. 2023, pp. 1–7.

[7] S. Shakibhamedan, N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini, and N. TaheriNejad, "ACE-CNN: Approximate carry disregard multipliers for energy-efficient CNN-based image classification," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 5, pp. 2280–2293, May 2024.

[8] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, "Approximate computing methods for embedded machine learning," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 845–848.

[9] S. Shakibhamedan, A. Aminifar, L. Vassallo, and N. TaheriNejad, "Harnessing approximate computing for machine learning," in *Proc. IEEE Comput. Soc. Annu. Symp.*, Jun. 2024, pp. 1–6.

[10] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 1–34, Aug. 2017, doi: 10.1145/3094124.

[11] C. Ossimitz and N. TaheriNejad, "A fast line segment detector using approximate computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[12] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.

[13] S. E. Fatemieh, S. S. Farahani, and M. R. Reshadinezhad, "LAHAF: Low-power, area-efficient, and high-performance approximate full adder based on static CMOS," *Sustain. Comput. Informat. Syst.*, vol. 30, Jun. 2021, Art. no. 100529.

[14] H. A. Almurib, T. N. Kumar, and F. Lombardi, "Inexact designs for approximate low power addition by cell replacement," in *Proc. Design, Autom. Test Eur. Conf. Exhibition*, 2016, pp. 660–665.

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

[16] 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, pp. 6–873, Aug. 2010.

[17] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, Aug. 2009, pp. 33–36, doi: 10.1109/NANOARCH.2009.5226356.

[18] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008, doi: 10.1038/nature06932.

[19] 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, Sep. 2018.

[20] N. TaheriNejad, "SIXOR: Single-cycle in-memristor XOR," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 5, pp. 925–935, May 2021.

[21] K. A. Ali, "New design approaches for flexible architectures and in-memory computing based on memristor technologies," Ph.D. thesis, Dept. Electronique, IMT Atlantique-ELEC, Nantes, France, 2020. [Online]. Available: https://theses.hal.science/tel-03134905

[22] D. Radakovits and N. Taherinejad, "BEhavioral leakage and IntEr-cycle variability emulator model for ReRAMs," 2021, *arXiv:2103.04179*.

[23] C. Li et al., "In-memory computing with memristor arrays," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2018, pp. 1–4.

[24] S. Ganjeheizadeh Rohani, N. Taherinejad, and D. Radakovits, "A semi-parallel full-adder in IMPLY logic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 297–301, Jan. 2020.

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

[26] 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.

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

[28] 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.

[29] S. E. Fatemieh, M. R. Reshadinezhad, and N. TaheriNejad, "Approximate in-memory computing using memristive IMPLY logic and its application to image processing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 3115–3119.

[30] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Memristor-based IMPLY logic design procedure," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, Oct. 2011, pp. 142–147, doi: 10.1109/ICCD.2011.6081389.

[31] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Aug. 2018, pp. 1–7.

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

[33] P. Huang et al., "Reconfigurable nonvolatile logic operations in resistance switching crossbar array for large-scale circuits," *Adv. Mater.*, vol. 28, no. 44, pp. 9758–9764, Nov. 2016, doi: 10.1002/adma.201602418.

[34] S. Asgari, M. R. Reshadinezhad, and S. E. Fatemieh, "Energy-efficient and fast IMPLY-based approximate full adder applying NAND gates for image processing," *Comput. Electr. Eng.*, vol. 113, Jan. 2024, Art. no. 109053.

[35] S. E. Fatemieh, M. R. Reshadinezhad, and N. TaheriNejad, "Fast and compact serial IMPLY-based approximate full adders applied in image processing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 175–188, Mar. 2023.

[36] 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.

[37] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surveys*, vol. 48, no. 4, pp. 1–33, Mar. 2016.

[38] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020.

[39] F. Sabetzadeh, M. H. Moaiyeri, and M. Ahmadinejad, "A majority-based imprecise multiplier for ultra-efficient approximate image multiplication," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4200–4208, Nov. 2019, doi: 10.1109/TCSI.2019. 2918241.

[40] W. Zhou, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, May 2004.

[41] G.-H. Chen, C.-L. Yang, L.-M. Po, and S.-L. Xie, "Edge-based structural similarity for image quality assessment," in *IEEE Int. Conf. Acoust. Speed Signal Process. Proc.*, Jul. 2006, pp. 1–12.

[42] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing," in *Proc. 13th IEEE Int. Conf. Nanotechnol.*, Aug. 2013, pp. 690–693.

[43] S. E. Fatemieh and M. R. Reshadinezhad, "Power-efficient, high-PSNR approximate full adder applied in error-resilient computations based on CNTFETs," in *Proc. 20th Int. Symp. Comput. Archit. Digit. Syst. (CADS)*, Aug. 2020, pp. 1–5.

[44] Y. S. Mehrabani, R. F. Mirzaee, Z. Zareei, and S. M. Daryabari, "A novel high-speed, low-power CNTFET-based inexact full adder cell for image processing application of motion detector," *J. Circuits, Syst. Comput.*, vol. 26, no. 5, May 2017, Art. no. 1750082.

[45] S. Muthulakshmi, C. S. Dash, and S. R. S. Prabaharan, "Memristor augmented approximate adders and subtractors for image processing applications: An approach," *Int. J. Electron. Commun.*, vol. 91, pp. 91–102, Jul. 2018.

[46] S. Muthulakshmi, C. S. Dash, and S. R. S. Prabaharan, "Memristor-based approximate adders for error resilient applications," in *Nanoelectronic Materials and Devices*. Cham, Switzerland: Springer, 2018, pp. 51–59.

[47] K. Bickerstaff and E. E. Swartzlander, "Memristor-based arithmetic," in *Proc. Conf. Rec. Forty 4th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2010, pp. 1173–1177.

[48] D. Radakovits, M. Jungwirth, S. M. Laube, and N. TaheriNejad. (Sep. 2019). *Second (V2.0) LTSpice Implementation of VTEAM*. [Online]. Available: https://www.ict.tuwien.ac.at/staff/taherinejad/projects/memristor/files/vteam2.asc

[49] (2017). *Knowm*. [Online]. Available: https://knowm.org

[50] F. Seiler. (2024). [Online]. Available: https://github.com/fabianseiler/Ax-Image-Processing

[51] A. Prati, I. Mikic, M. M. Trivedi, and R. Cucchiara, "Detecting moving shadows: Algorithms and evaluation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 7, pp. 918–923, Jul. 2003.

[52] (2017). *USC University of Southern California, Signal and Image Processing Institute (SIPI)*. [Online]. Available: https://sipi.usc.edu/database/

[53] N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini, and N. TaheriNejad, "Carry disregard approximate multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 12, pp. 4840–4853, Dec. 2023.

[54] E. Zacharelos, I. Nunziata, G. Saggese, A. G. M. Strollo, and E. Napoli, "Approximate recursive multipliers using low power building blocks," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 3, pp. 1315–1330, Jul. 2022.

Fabian Seiler is currently pursuing the M.Sc. degree in embedded systems with TU Wien, Vienna, Austria. He wrote his B.Sc. thesis about approximate memristive in-memory computing. He has published a conference paper about approximated memristive adders. His main research interests are memristive circuits and systems, approximate computing, and embedded systems.

Nima TaheriNejad (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from The University of British Columbia (UBC), Vancouver, Canada, in 2015. He is currently a Full Professor with Heidelberg University, Heidelberg, Germany, and affiliated with TU Wien (formerly known also as Vienna University of Technology), Vienna, Austria. He has published three books, three patents, and more than 90 articles. His areas of work include in-memory computing, cyber-physical and embedded systems, systems on chip, memristor-based circuit and systems, self-systems, and health-care. He has served as a reviewer and an editor of many journals and conferences. He has also been an organizer and the chair of various conferences and workshops. He has received several awards and scholarships from universities, conferences, and competitions, he has attended. This includes the Best University Booth Award at DATE 2021, First Prize in the 15th Digilent Design Contest (2019) and in the Open-Source Hardware Competition at Eurolab4HPC (2019), and Best Teacher and Best Course Award at TU Wien (2020).