Efficient Image Processing via Memristive-Based Approximate In-Memory Computing

Fabian Seiler and Nima TaheriNejad, *Member, IEEE*

***Abstract*—Image processing algorithms continue to demand higher performance from computers. However, computer performance is not improving at the same rate as before. In response to the current challenges in enhancing computing performance, a wave of new technologies and computing paradigms is surfacing. Among these, memristors stand out as one of the most promising components due to their technological prospects and low power consumption. With efficient data storage capabilities and their ability to directly perform logical operations within the memory, they are well-suited for in-memory computation (IMC). Approximate computing emerges as another promising paradigm, offering improved performance metrics, notably speed. The tradeoff for this gain is the reduction of accuracy. In this article, we are using the stateful logic material implication (IMPLY) in the semi-serial topology and combine both the paradigms to further enhance the computational performance. We present three novel approximated adders that drastically improve speed and energy consumption with an normalized mean error distance (NMED) lower than 0.02 for most scenarios. We evaluated partially approximated Ripple carry adder (RCA) at the circuit-level and compared them to the State-of-the-Art (SoA). The proposed adders are applied in different image processing applications and the quality metrics are calculated. While maintaining acceptable quality, our approach achieves significant energy savings of 6%–38% and reduces the delay (number of computation cycles) by 5%–35%, demonstrating notable efficiency compared to exact calculations.**

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

I. Introduction

With the rising demand for image processing applications in various fields, more processing power has to be allocated to these tasks. Since, the required image quality and time to process these applications is also increasing drastically, current technology is facing serious challenges in keeping up with the demand. In addition to this, the enhancement of general-purpose computing performance is stagnating with challenges, such as the slowdown of Moore's law [1] and the von Neumann bottleneck. Hence, nowadays considerable attention is directed toward exploring novel technologies and computing paradigms in this domain. in-memory computation (IMC) represents a methodology for performing computations directly within memory, offering a potential solution to circumvent the von Neumann bottleneck that typically occurs between the logic and memory. Among the notable emerging technologies, the memristor stands out as a promising candidate. The compelling attributes of low power consumption and a compact form factor, as highlighted by Williams [2], position memristor technology as one of the most likely candidates for future computing advances. With the ability to store data nonvolatile through its resistive state and the ability to perform logical operations, it is ideally suited as a memory cell [3], [4]. In the realm of IMC, the stateful logic material implication (IMPLY) proves to be a favorable choice; its well-established and widely recognized nature, coupled with compatibility with the crossbar array, positions it as an ideal candidate for such applications [3], [5]. It is also the most reliable when compared to the other stateful memristive logics [6]. The currently available structures to perform IMPLY operations with, can be divided into serial, parallel, and hybrid topologies [7], [8], [9], [10]. A hybrid structure, such as the semi-serial topology combines the advantages of the serial and parallel approach and so offers a more efficient approach [8].

An upcoming computer paradigm that is a possible solution to the power-wall problem is the approximation of computational processes [1], [11]. The adoption of approximate computing leads to improved performance metrics, such as speed, area, and energy consumption, which all would benefit image processing applications. The tradeoff for these enhancements is the reduction of the accuracy of these computations [1], [11], [12]. Since, image and video processing applications are of error-resilient nature, the approximation of some part of the process could lead to stark gains in computing time and power consumption [1], [13]. Other important fields, such as machine learning, pattern recognition, communication, data mining, and robotics are often in someway connected to imaging applications and would also benefit [1], [13], [14], [15].

Addition operations are fundamental elements in digital arithmetic, given that a substantial portion of basic instructions relies on the addition and multiplication [12]. The efficiency of the associated half and full adders significantly influences the overall performance of the computational process. In this work, we extend on the approximated adder from [16] and present three novel adder algorithms in the semi-serial IMPLY-based topology to complete this methodological approach.

Manuscript received 26 July 2024; accepted 28 July 2024. Date of current version 6 November 2024. This article was presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Fabian Seiler.)*

Fabian Seiler is with Technische Universität Wien (TU Wien), 1040 Wien, Austria (e-mail: fabian.seiler@student.tuwien.ac.at).

Nima TaheriNejad is with Heidelberg University, 69117 Heidelberg, Germany, and also with Technische Universität Wien (TU Wien), 1040 Wien, Austria (e-mail: nima.taherinejad@ziti.uni-heidelberg.de).

Digital Object Identifier 10.1109/TCAD.2024.3438113

The algorithms use an approximated approach to create an inexact truth table. The number of memristors, the hardware complexity, and the power consumption were drastically reduced if compared to the exact semi-serial algorithm [8]. The primary advancement compared to the State-of-the-Art (SoA) lies in the notable reduction of steps required per bit. To our knowledge, we present the fastest IMPLY-based approximate adder algorithms. With our approach, we are able to drastically reduce both time and energy requirements for basic image processing applications with only a marginal loss of quality that can be considered negligible for the human visual system. With our memristor-based approach, scalability and performance gains have a lot more potential for the increasing demands of image processing applications than the complementary metal-oxide semiconductor (CMOS) era.

This work is divided into seven sections. In Section II we cover the necessary background and review key papers in related areas. The methodology for designing the algorithms and their exact operation is described in Section III. In Section IV we simulated the adders at the circuit-level, verified their functionality, and evaluated the error analysis using the standard metrics. We compared to the other exact and approximated algorithms in Section V. We simulated three image processing applications and evaluated the quality of the outcomes. The results of these can be seen in Section VI, where we also discuss the gains on the application-level In Section VII we conclude this article and discuss future work.

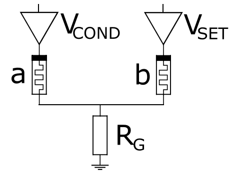
II. Background

A. Memristors

The memristor was originally discovered by Chua [17] and physically realized by Strukov et al. [18]. The memristor complements the absent symmetry in representing the four fundamental passive electronic components, alongside the resistor, capacitor, and inductor [17]. With its resistive states enabling nonvolatile data storage, it establishes itself as the optimal component for a memory cell [3], [4]. Other advantages of the memristor include low power consumption, as well as low write time and small dimension of the device [19], [20], [21]. The minimum (R_{on}) and maximum (R_{off}) resistance values of the memristor are set by the applied voltage and the direction of current flow, forming a hysteresis curve. Conventionally, we can assume the minimum resistance value is equivalent to a logical "1" and the maximum resistance value equal to a logical "0" [7], [22], [23].

B. In-Memristor Logic - IMPLY

Memristor-based IMPLY is a stateful logic with memristors that has the advantage that no reads and writes are required to perform logical operations [20]. IMPLY was introduced by Hewlett Packard (HP), which established itself as the first stateful logic [3], [9], [24]. There exist other stateful logic forms for memristors, such as FELIX [25], SIXOR [20], MAGIC [26], and TSML [27] as well as nonstateful logic as MRL [28]. However, in this work we focus on IMPLY, as it is the most reliable stateful logic [6] and the only one where approximations have been presented [16], [29], [30].



(a)

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

(b)

Fig. 1. IMPLY operation [3]. (a) Gate structure. (b) Truth table.

The basic structure to perform IMPLY operations is shown in Fig. 1(a). Two memristors are used, to which different voltages V_{COND} and V_{SET} can be applied. The two memristors are connected to a resistor which needs to fulfill the requirement $R_{\mathrm{on}} << R_G << R_{\mathrm{off}}$. The two applied voltages must also satisfy the condition in $V_{\mathrm{COND}} < V_C < V_{\mathrm{SET}}$ for the IMPLY logic to be possible, where V_C is the threshold voltage of the memristor [3], [9], [19], [24], [31]. An IMPLY operation is represented by $a \to b$, where the logic inputs correspond to the resistive state of the memristors. To perform $a \to b$, a short pulse of V_{COND} and V_{SET} is applied [3], [24]. In this process, the b-memristor loses its previous state and the result of this operation is stored in it instead. The truth table of this operation can be found in Fig. 1(b).

C. IMPLY-Based Full Adders

Adders based on the IMPLY logic can be divided into three categories: 1) serial; 2) parallel; and 3) hybrid forms, such as semi-serial or semi-parallel. In the serial structure, memristors are placed in the same row or column of a crossbar array as in [3], [4], and [7]. The best serial algorithm needs $22n$ steps and $2n+3$ memristors for an n-bit calculation [7]. The parallel structure consists of individual rows that are not contiguous, so calculations can be performed in parallel [7], [9], [19]. Since, the individual bits are dependent on the calculation of their predecessor, not all steps can be parallelized and must therefore be processed sequentially. The full adder algorithm from [19] requires $5n + 16$ steps and $4n + 1$ memristors and n external switches for the n-bits. In the semi-parallel full adder, the serial structure is divided into two rows with one input and a work memristor per row [10]. If two operations can be performed in parallel, it is possible in this structure which leads to it only requiring $17n$ steps and $2n + 3$ memristors as well as three switches for the n-bit. The semi-serial structure shown in Fig. 2 is a hybrid structure that achieves a better balance between the space consumption and speed compared to the serial and parallel [22]. This topology consists of two parallel rows with the inputs, which can connect to four work memristors, c_{in}, and c-memristor. This totals to $2n + 6$ memristors and 12 switches. The exact algorithm from [22] requires $10n + 2$ steps for the n-bit.

D. Approximate Computing

The fundamental approach to approximate computing involves redefining logic by eliminating gates or individual transistors and formulating a new truth table. With this approximation, performance metrics, such as energy consumption,

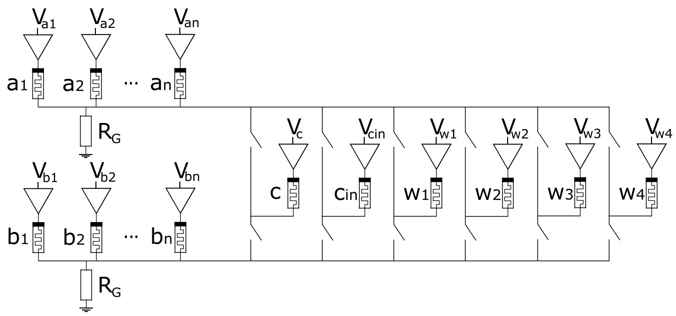


Fig. 2. IMPLY-based semi-serial n-bit adder structure [22].

area usage, and processing time are significantly reduced. The accuracy of the calculation is reduced as a tradeoff. To evaluate the degree of inaccuracy, error metrics were used in SoA publications, such as [14], [32], [33], [34], [35], and [36]. The most important and used metrics in this work are error distance (ED), error rate (ER), relative error distance (RED), mean error distance (MED), normalized mean error distance (NMED), and mean relative error distance (MRED). One application of approximated computing is image processing, since it has a high error resistance [11], [32]. A common quality metric is the peak signal-to-noise ratio (PSNR) which indicates how strong the noise is compared to the actual signal. A value of more than 30 dB is considered acceptable [37], [38]. Especially for images, the structural context is relevant for the human visual system [39]. Therefore, two more quality metrics, structural similarity index measure (SSIM) and mean structural similarity index measure (MSSIM) are often used in image processing [39], [40]. Many variants of approximated CMOS-based full adder have been published, all of which have used different approximation methods, such as [11], [12], [32], and [34]. Other technologies have also been used to achieve a better approximation [35], [41].

E. Approximate In-Memristor Computing

Approximated full adders based on the memristors have recently been proposed. In [42] and [43], they utilized the memristor ratioed logic (MRL) from [28] and changed the truth table of the full adder to save memristors. The main disadvantage of MRL is that the additional CMOS-inverter and amplifier are required. Their approximate design reduced the number of required memristors from 33 to 10 and some CMOS inverters and evaluated the adders with image addition. Approximated full adders that utilized IMPLY have also been presented in [23], [29], and [30], where new approximate algorithms for the serial structure were proposed. They simplified the truth table and utilized specific input vectors to minimize the number of required steps. Thereby they reduced energy consumption by up to 68% and the number of required steps by up to 42% and evaluated their adders in different image processing applications. Seiler and TaheriNejad [16] presented an approximated adder in the semi-serial topology that utilizes the similarity Sum $\approx \overline{C_{\text{out}}}$. Here, we propose three new algorithms for the semi-serial adders with a more advanced design methodology that advances the approach from [16] and results in better performances in different aspects. We compare our results with the SoA and present the results in Section V.

III. Proposed Approximate Full Adders

A. Methodology

In our work, the method to design approximated circuits is to take the correct logic as a reference and derive approximated logic from it. Typically, this is done by either changing or omitting components or using a modified truth table so that the speed and/or the number of components required can be reduced [14], [32], [33]. As we are working on the IMPLY-based semi-serial structure, we use only IMPLY and false operations. Together they form a complete logic set with which we can emulate the Boolean logic [7], [44]. It turns out that an inversion needs only one IMPLY operation and only OR and NAND need two IMPLY operations. Therefore, the approximations focus on using these operations and false to reduce the required steps [16]. We developed the approximations in this work by introducing an intentional error in the truth table of an exact full adder at one place of C_{out}. For each case, we determined the conjunctive and disjunctive normal forms using the Karnough-Veigh-diagramm (KVD) and verified them to be representable in as few steps as possible in IMPLY logic. Operating within the semi-serial structure detailed in [8], we capitalize on its built-in parallelization capability. This empowers us to concurrently compute numerous essential steps, resulting in significant time savings in computational processes. We exclusively employed logical approximations that align seamlessly with the efficient representation enabled by this parallelization approach. In each of the presented algorithms, we represented the sum of the full adder as Sum $\approx \overline{C_{\text{out}}}$. We are using this approach because it exploits the similarities and requires only one additional computational step (inversion) to calculate the Sum, based on the approach from [16]. To ensure that the algorithm of the approximated full adder is compatible with the algorithm of the exact full adder from [8], we took care in this work that the calculated Sum is always stored in the respective a-memristor and the carry bit is stored in the c-memristor. We have chosen to include a shortened description of the algorithm from [16] in Section III-D to give the reader the complete picture of the entire space of this methodological approach which this algorithm is a part of. We labeled the approximated algorithms in the following sections based on the placement of the error in the truth table.

B. Approximated Algorithm 1

In this algorithm, we introduce an error in the truth table in the case $[a,\ b,\ c]$ = "001" which results in C_{out} having an ER of (1/8). Since, the sum is equal to the inverted carry-out, it has an ER of (3/8) since in the cases $[a,\ b,\ c]$ = "000" and $[a,\ b,\ c]$ = "111," and Sum is not equal to the inverse of C_{out}

$$C_{\text{out}} = ab + c = \left(a \rightarrow \overline{b}\right) \rightarrow c \tag{1}$$

$$\text{Sum} = \overline{ab + c} = \overline{\left(a \rightarrow \overline{b}\right) \rightarrow c}. \tag{2}$$

TABLE I
APPROXIMATED ALGORITHM 1

Steps	Section 1	Section 2	Equivalent Logic
-		$w_1 = 0$	$False(w_1)$
1		$w_1^{'} = b \to w_1$	$w_1 = \bar{b}$
2	$w_1^{''} = a \to {w_1}'$		$w_1 = a \to \bar{b}$
3	$a = 0$	$c' = w_1^{''} \to c$	$False(a)$, $c = (a \to \bar{b}) \to c = Cout$
4	$a^{'} = c^{'} \to a$	$w_1 = 0$	$a = \overline{(a \to \bar{b})} \to c = Sum$, $False(w_1)$

TABLE II
APPROXIMATED ALGORITHM 2

Steps	Section 1	Section 2	Equivalent Logic
-		$w_1 = w_2 = 0$	$False(w_1, w_2)$
1	$w_1^{'} = c \to w_1$	$w_2^{'} = b \to w_2$	$w_1 = \bar{c}, w_2 = \bar{b}$
2	$w_1^{''} = a \to w_1^{'}$	$c = 0$	$w_1 = a \to \bar{c}, False(c)$
3	$c^{'} = w_1^{''} \to c$		$c = \overline{a \to \bar{c}}$
4	$a = 0$	$c^{''} = w_2^{'} \to c^{'}$	$False(a)$, $c = \bar{b} \to (\overline{a \to \bar{c}}) = Cout$
5	$a^{'} = c^{''} \to a$	$w_1 = w_2 = 0$	$a = \overline{\bar{b} \to (\overline{a \to \bar{c}})} = Sum$, $False(w_1, w_2)$

TABLE III
APPROXIMATED ALGORITHM 3 [16]

Steps	Section 1	Section 2	Equivalent Logic
-		$w_1 = w_2 = 0$	$False(w_1, w_2)$
1	$w_2^{'} = a \to w_2$	$w_1^{'} = c \to w_1$	$w_2 = \bar{a}$, $w_1 = \bar{c}$
2	$c = 0$	$w_1^{''} = b \to w_1^{'}$	$False(c)$, $w_1 = b \to \bar{c}$
3	$c^{'} = w_1^{''} \to c$		$c = \overline{b \to \bar{c}}$
4	$a = 0$	$c^{''} = w_2^{'} \to c^{'}$	$False(a)$, $c = \bar{a} \to (\overline{b \to \bar{c}}) = Cout$
5	$a^{'} = c^{''} \to a$	$w_1 = w_2 = 0$	$a = \overline{\bar{a} \to (\overline{b \to \bar{c}})} = Sum$, $False(w_1, w_2)$

TABLE IV
APPROXIMATED ALGORITHM 4

Steps	Section 1	Section 2	Equivalent Logic
-		$w_1 = w_2 = 0$	$False(w_1, w_2)$
1	$w_1^{'} = a \to w_1$	$w_2^{'} = c \to w_2$	$w_1 = \bar{a}, w_2 = \bar{b}$
2	$c = 0$	$b^{'} = w_1^{'} \to b$	$False(c), b = \bar{a} \to b$
3		$w_2^{''} = b^{'} \to w_2^{'}$	$w_2 = (\bar{a} \to b) \to \bar{c}$
4	$a = 0$	$c^{'} = w_2^{''} \to c$	$False(a), c = \overline{(\bar{a} \to b) \to \bar{c}} = Cout$
5	$a^{'} = c^{'} \to a$	$w_1 = w_2 = 0$	$a = (\bar{a} \to b) \to \bar{c} = Sum, False(w_1, w_2)$

In (1) and (2) the logical functions of C_{out} and Sum can be seen in the Boolean and IMPLY logic form. We took advantage of the fact that for an OR operation in IMPLY logic, one of the inputs must be inverted. Therefore, the NAND operation can be used directly and thus calculation steps can be omitted. Since, C_{out} can be stored directly in the c-memristor, we only need three steps for its calculation and another one for the storage of Sum in the a-memristor. Before starting the calculation, a false operation is required once on the work memristor, which can be executed in parallel during the repetitions of the algorithm in the fourth step. The exact process of the algorithm can be seen in Table I. It requires only $4n + 1$ steps and $2n + 2$ memristors for the n-bits addition.

C. Approximated Algorithm 2

In the second algorithm, we introduced an error in the third row of the truth table due to the approximation, which leads to a C_{out} of 1 for the case $[a,\ b,\ c]$ = "010." C_{out} has an ER of (1/8). Since, in this approximation Sum $= \overline{C_{\text{out}}}$ it follows that the Sum has an ER of (3/8) since again the least-significant bit (LSB) and most significant bit (MSB) are incorrect:

$$C_{\text{out}} = ac + b = \bar{b} \to \overline{(a \to \bar{c})} \tag{3}$$

$$\text{Sum} = \overline{ac + b} = \overline{\bar{b} \to \overline{(a \to \bar{c})}}. \tag{4}$$

We used the logical functions in (3) and (4). We saved steps given that first NAND and then OR are executed. With this procedure, steps can be combined in IMPLY form. Since, proper storage of C_{out} in the c-memristor is necessary, $\bar{c}$ is first stored in a work memristor, and False(c) was applied so that the c-memristor is available to store the inversion of $a \to \bar{c}$. To comply with the default memory location two more inversions are necessary, resulting in this algorithm requiring $5n+1$ steps and $2n + 3$ memristors at the n-bits. The exact flow of the algorithm can be seen in Table II.

D. Approximated Algorithm 3 [16]

This algorithm was already explained in more detail in [16]. The shortened version is included here since it is also part of the design approach we implemented in this work and to show the symmetry of the algorithm presented in [16] with respect to the second algorithm in this article. The error placement of C_{out} for this algorithm lies at $[a,\ b,\ c]$ = "100." This reduces the truth table to a form where the first three rows are 0 and after that, all the entries are logical 1. This leads to C_{out} having an ER of (1/8) and Sum having an ER of (3/8)

$$\text{Cout} = bc + a = \bar{a} \to \overline{(b \to \bar{c})} \tag{5}$$

$$\text{Sum} = \overline{bc + a} = \overline{\bar{a} \to \overline{(b \to \bar{c})}}. \tag{6}$$

In (5) and (6), the logical function corresponding to the approximation can be seen [16]. This approximation is a symmetrical approach to the second algorithm we proposed in Section III-C, with only the inputs a and b swapped. The exact procedure can be seen in Table III, where we can see that it also requires $5n+1$ steps and $2n+3$ memristors for an n-bit calculation.

E. Approximated Algorithm 4

We changed the truth table of this algorithm at $[a,\ b,\ c]$ = "110," so that in this case C_{out} is equal to 0. The truth table can be seen in Table V, where the red marked bits represent the errors introduced by us. It can be seen that C_{out} again has an ER of (1/8) and Sum has an ER of (3/8)

$$\text{Cout} = (a + b)c = \overline{(\bar{a} \to b) \to \bar{c}} \tag{7}$$

$$\text{Sum} = \overline{(a + b)c} = (\bar{a} \to b) \to \bar{c}. \tag{8}$$

In this algorithm, we first perform an OR operation and then an NAND operation, which is not possible otherwise due to the selected memory locations. Equations (7) and (8) show this algorithm's logical functions we created and the reason for the necessity to perform a double inversion. The exact procedure can be found in Table IV. It should be noted that *False*(a) could also be performed in steps 2 or 3. We selected and implemented the chosen variant due to its superior energy efficiency observed during the circuit simulations with an equal number of steps. This algorithm requires $5n + 1$ steps and $2n + 3$ memristors for a calculation of the n-bits.

TABLE V
TRUTH TABLE OF THE PRESENTED ALGORITHMS WITH ERRONEOUS PLACES MARKED IN RED

Inputs			Exact		Algorithm 1		Algorithm 2		Algorithm 3 [16]		Algorithm 4	
a	b	c	Sum	Cout	Sum	Cout	Sum	Cout	Sum	Cout	Sum	Cout
0	0	0	0	0	1	0	1	0	1	0	1	0
0	0	1	1	0	0	1	1	0	1	0	1	0
0	1	0	1	0	1	0	0	1	1	0	1	0
0	1	1	0	1	0	1	0	1	0	1	0	1
1	0	0	1	0	1	0	1	0	0	1	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1
1	1	0	0	1	0	1	0	1	0	1	1	0
1	1	1	1	1	0	1	0	1	0	1	0	1

TABLE VI
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 VII
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

IV. CIRCUIT-LEVEL SIMULATION AND ERROR METRICS

A. Simulation Setup

To simulate the proposed approximated full adders we used a model based on the voltage-controlled threshold adaptive memristor (VTEAM) model [31], which is implemented in SPICE and fitted to measurement data [8], [45]. We used LT-SPICE to perform these simulations to confirm the correct functionality and verify it for every input combination. The parameters we selected are listed in Table VI. It is important to highlight that the specified parameters are outcomes derived from tailoring the model to the real devices, in this case discrete known memristors [46]. Like with the difference between the discrete and integrated CMOS devices, this leads to slower operations and increased power consumption. It is important to recognize that while these outcomes reflect the adaptation of the model to discrete memristors, integrated memristors offer significant improvements in operational speed and power efficiency. However, since we do not have access to integrated memristors, to ensure relevant and realistic implementability of our proposed circuits, we use measurement fitted models mentioned above. We note that IMPLY has been experimentally validated in [3]. The specific parameters of the IMPLY logic that we used in this simulation are listed in Table VII. The parameters were chosen following the same setup already used in [7], [16], [19], [29], and [30]. This allows for a good comparison to existing approximated and exact full adder.

Real memristors show nonideal behaviors, one of the most important of which is their resistance variation, where a deviation of R_{on} and R_{off} has to be expected. To encompass this in our experiments, we repeated our simulations where the low and high resistive states of the memristors deviate. We evaluated the resulting state for Sum and C_{out} at the end of each algorithm for each possible input combination. The range that the resulting states can assume is illustrated in Fig. 3. The results are correct and within the 33% threshold for up to ±30% deviation range. Even with a deviation range of 50%, only three bit flips occur (the fourth algorithm), underlining the reliability of the proposed solutions. We presented this state deviation as shaded areas in the following waveform figures.

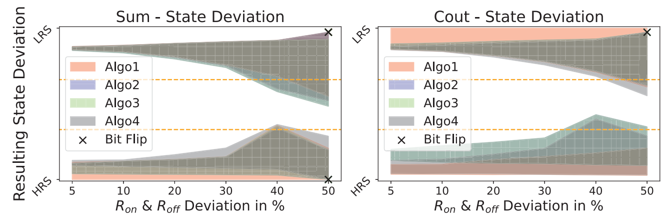


Fig. 3. Resulting states (Sum and Cout) deviation with varying R_{on} and R_{off}. Orange lines mark the 33% thresholds.

B. Simulation Results

To verify that each algorithm operates correctly with the mentioned SoA parameters, we simulated them using LT-SPICE. We took the semi-serial structure from [8] and tested all the possible input combinations that can occur for functionality. The input states of the *a*, *b*, and *c*-memristor were set before the algorithm was executed. To accommodate the presented algorithms we included the step that resets all the work memristors in every algorithm. This step will be parallelized after the first iteration as explained in more detail in Section III. The function is considered correct if, after the conclusion of the algorithm, both the Sum and C_{out} align with the solutions specified in the corresponding truth table. As specified in Table VII, we let each step of the algorithms last 30 µs. In the second, third, and fourth algorithms, the C_{out} is calculated at the fourth step which corresponds to the time between 120 $\mu s - 150$ µs. Since, the first algorithm's logic function allows for a better representation with IMPLY logic, the calculation of the carry-out is done in the third step. This corresponds to the period between 90 $\mu s - 120$ µs. For all the algorithms the C_{out} is stored in the *c*-memristor to allow for a flawless continuation with iterations. The calculation of Sum $\approx \overline{C_{\mathrm{out}}}$ is done in the period between 120 $\mu s - 150$ µs for the first algorithm which is the fourth step. For the other algorithms, this calculation is done in the fifth step in the period of 150 $\mu s - 180$ µs. We used the convention of always saving the Sum result in the *a*-memristor of the corresponding bit for all the presented algorithms. This saving scheme was also applied at the third algorithm in [16]. We examined the simulation of each algorithm for all the eight input possibilities, and the expected exact and erroneous outputs agreed with the corresponding truth tables from Section III.

The output waveform of each memristor of Algorithm 1 was plotted at cases "AinBinCin" = 100 and 001 to show a correct calculation of Sum and C_{out} in the first case and a calculation showing the intentional error produced by our chosen approximation. We present the first case with the correct outputs in Fig. 4(a) and the case with the error in Fig. 4(b).

To ensure correct functionality of the full adder at the circuit-level with multiple bits, we tested all the algorithms as 4-bit Ripple carry adder (RCA). For this, we let the lowest two

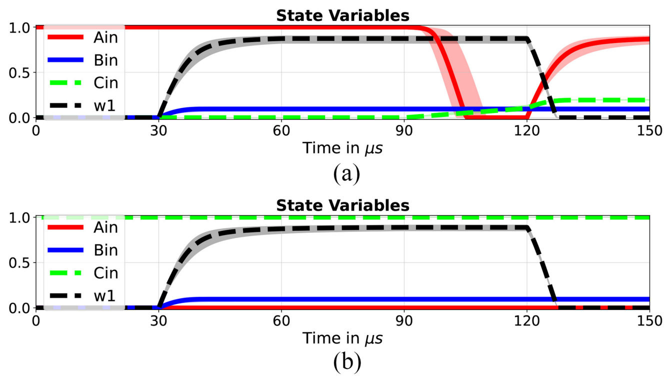


Fig. 4. Two example simulations of Algorithm 1, illustrating the resistive deviation of ±20% as shaded areas. (a) AinBinCin = 100 with correct output. (b) AinBinCin = 001 with approximated (erroneous by design) output.

bits use the proposed algorithm and the higher two bits use the exact full adder algorithm for a semi-serial topology from [8]. We simulated this procedure for all the presented algorithms. For each algorithm presented in this work, five random pairs of numbers were added by our LT-SPICE simulation and the results agree with our theoretical calculations.

C. Error Analysis

1) Error Metrics for 8-Bit RCA: To compare the erroneous behavior of the approximated full adders presented in this work, we use the error metrics introduced in Section II. The exact definition of MED, NMED, and MRED can be found in

$$\text{MED} = \frac{1}{2^{2n}} \cdot \sum_{i=1}^{2^{2n}} |\text{SUM}_{\text{Exact}} - \text{SUM}_{Ax}|_i \tag{9}$$

$$\text{NMED} = \frac{\text{MED}}{2^{n+1} - 1} \tag{10}$$

$$\text{MRED} = \frac{1}{2^{2n}} \cdot \sum_{i=1}^{2^{2n}} \frac{|\text{SUM}_{\text{Exact}} - \text{SUM}_{Ax}|_i}{\text{SUM}_{\text{Exact},i}}. \tag{11}$$

More detailed information about these metrics can be found in [32], [33], [34], [35], and [36]. We performed the following simulations in MATLAB with Cin = 0. Therefore, we created a behavioral-level model of the RCA, which is variable for the respective approximation degree and the number of maximum bits. For the 8-bit case, we applied all 65 536 input combinations to the RCAs with different approximation degrees. With this setup the MED, NMED, and MRED were determined. We used the approximated full adders as the LSBs of a RCA structure. The cases with one to five approximated full adders were recorded in Table VIII. We observed that the MED and thus also the NMED roughly double per included approximated full adder. It is noticeable that the second and third [16] algorithms have the same results for the error metrics at 8-bit. This result is expected since the truth tables of the two algorithms are identical when the inputs a and b are swapped. The second and third [16] algorithms give the best results for MED, NMED, and MRED compared to the other two. The fourth algorithm gives the worst results, which are up to 16% worse than those of the other algorithms.

Full Adder

2) Error Metrics for 16-Bit and 32-Bit RCA: In the analysis of 16- and 32-bit RCA, we used one million randomly generated numbers as input variables. We did this because for a complete evaluation 2^{2n} input combinations would be needed, which is computationally intensive. We again performed a behavioral-level simulation in MATLAB and calculated MED, NMED, and MRED. Therefore, we used a RCA structure and increased the number of approximated adders. The approximated adders are again calculating the lower bits and the approximation degrees indicate the number of approximated full adders from the total number of full adders. The 16- and 32-bit simulations yield drastically lower NMED and MRED for the lower approximation degrees in comparison to the 8-bit simulation. When only approximated adders are used the quality metrics of the different bit simulations are almost equal. This indicates that an approximated full adder generates a substantially higher quality output with a higher number of bits. The second and third [16] algorithms would again give the same results if all the input possibilities were fully simulated. Since, only one million input combinations were validated, the results are subject to stochastic deviations. Nevertheless, the present figures should be a reliable representation, given that the one million input combinations were chosen randomly. The same is true for all the error metrics of the 16- and 32-bit cases. It is noticeable that the second and third [16] algorithms perform better than the other two we presented. The first and fourth algorithms produce similar results for the 16- and 32-bit error metrics. We displayed the results in Table VIII.

V. Circuit-Level Comparison

We compared the algorithms presented in this article and algorithm 3 from [16] with the exact full adders from [7], [8], [9], [10], and [19] and the approximated full adders from [20] and [30] in various circuit-level metrics.

A. Comparison With Exact Full Adders

1) Energy Consumption: We calculated the energy consumption with the LT-SPICE energy consumption tool for all the algorithms. Simulation encompassed all the feasible input combinations for a full adder. The result is defined as the mean value across all the simulations. Since, the first step of the algorithms is performed only before the first iteration, it is not considered in the results because it is negligible with respect to several bits. The energy consumption of the first algorithm is 28.8 pJ because only one work memristor was used. The energy consumption for the first step of the other three algorithms is 55.5 pJ. The formulas for all the presented adders in an RCA structure that embeds k approximated adders and n total adders are shown in

$$E_1(n,k) = 1.4509k + 3.8435(n-k) + 0.834 \tag{12}$$

$$E_2(n,k) = 1.6694k + 3.8435(n-k) + 0.865 \tag{13}$$

$$E_3(n,k) = 1.6678k + 3.8435(n-k) + 0.865 \tag{14}$$

$$E_4(n,k) = 1.8697k + 3.8435(n-k) + 0.865. \tag{15}$$

The energy consumption of the semi-serial topology [8] with the IMPLY specific values from Table VII was recreated

TABLE VIII
ERROR METRICS OF THE PRESENTED ALGORITHMS FOR THE 8/16/32-BIT RCA WITH VARYING APPROXIMATION DEGREES

Ax 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		
Algorithm 1	0.25	0.0005	0.0014	0.9056	0.0000069	0.000018	4.8151	$<$ e-09	$<$ e-08
Algorithm 2	0.5	0.0010	0.0028	1.1077	0.0000085	0.000024	4.5291	$<$ e-09	$<$ e-08
Algorithm 3*	0.5	0.0010	0.0028	1.1469	0.0000087	0.000026	4.3603	$<$ e-09	$<$ e-08
Algorithm 4	0.5	0.0010	0.0027	1.2503	0.0000095	0.000026	5.2720	$<$ e-09	$<$ e-08
	Approximation degree 2/8			Approximation degree 4/16			Approximation degree 8/32		
Algorithm 1	0.8750	0.0017	0.0049	4.6411	0.000035	0.000101	85.1028	$<$ e-08	$<$ e-07
Algorithm 2	1.1250	0.0022	0.0028	4.5378	0.000034	0.000096	71.7812	$<$ e-08	$<$ e-07
Algorithm 3*	1.1250	0.0022	0.0028	4.4258	0.000033	0.000098	71.8584	$<$ e-08	$<$ e-07
Algorithm 4	1.2500	0.0024	0.0069	5.2815	0.000040	0.000103	85.6307	$<$ e-08	$<$ e-07
	Approximation degree 3/8			Approximation degree 6/16			Approximation degree 12/32		
Algorithm 1	2.1562	0.0042	0.0122	20.0638	0.00015	0.00043	1374	$<$ e-06	$<$ e-06
Algorithm 2	2.2500	0.0044	0.0125	17.4923	0.00013	0.00037	1140	$<$ e-06	$<$ e-06
Algorithm 3*	2.2500	0.0044	0.0125	18.0559	0.00014	0.00038	1149	$<$ e-06	$<$ e-06
Algorithm 4	2.6250	0.0051	0.0146	21.4840	0.00016	0.00042	1363	$<$ e-06	$<$ e-06
	Approximation degree 4/8			Approximation degree 8/16			Approximation degree 16/32		
Algorithm 1	4.7266	0.0092	0.0273	84.0964	0.00064	0.0017	21865	0.0000025	0.0000072
Algorithm 2	4.4688	0.0087	0.0252	70.1606	0.00054	0.0014	17975	0.0000021	0.0000056
Algorithm 3*	4.4688	0.0087	0.0252	71.0475	0.00054	0.0015	18132	0.0000021	0.0000061
Algorithm 4	5.3125	0.0104	0.0299	85.3070	0.00065	0.0019	21786	0.0000025	0.0000071
	Approximation degree 5/8			Approximation degree 10/16			Approximation degree 20/32		
Algorithm 1	9.8887	0.0194	0.0589	330.6601	0.0025	0.0070	3.445e+05	0.000040	0.000113
Algorithm 2	8.9121	0.0174	0.0514	277.6172	0.0021	0.0061	2.875e+05	0.000033	0.000094
Algorithm 3*	8.9121	0.0174	0.0514	285.9959	0.0022	0.0062	2.943e+05	0.000034	0.000096
Algorithm 4	10.6562	0.0209	0.0616	341.0114	0.0026	0.0079	3.437e+05	0.000040	0.000111
	Approximation degree 8/8			Approximation degree 16/16			Approximation degree 32/32		
Algorithm 1	235.0758	0.4600	1.0019	66475	0.5072	0.9979	4.286e+09	0.4990	0.9999
Algorithm 2	203.3758	0.3980	0.9159	51123	0.3900	0.9152	3.436e+09	0.4000	0.9243
Algorithm 3*	203.3758	0.3980	0.9159	51626	0.3939	0.8665	3.527e+09	0.4105	0.8776
Algorithm 4	85.3320	0.1670	0.6281	22034	0.1681	0.6646	1.434e+09	0.1670	0.6405

* The error metrics for algorithm 3 were taken from [16]

TABLE IX
CIRCUIT-LEVEL COMPARISON TO EXACT SOA FULL ADDER

Full adder	**Energy consumption (nJ)**		**Improvement in comparison to [8]**	**No. of steps**		**Improvement in comparison to [8]**	**No. of memristors**		**No. of switches**
	n, k	n=8-bit	n=8-bit	n, k	n=8-bit	n=8-bit	n, k	n=8-bit	n, k
Serial Exact 1 [7]*	4.8250n	38.6000	-18%	22n	176	-115%	2n+3	19	0
Serial Exact 2 [19]*	4.0772n	32.6176	-3%	23n	184	-124%	2n+3	19	0
Parallel Exact [19]	-	-	-	5n+18	58	29%	4n+1	33	n
Semi-Parallel [10]*	4.8339n	38.6712	-18%	17n	136	-40%	2n+3	19	3
Semi-Serial Exact [8]*	3.8435n +0.8053	31.5533	-	10n+2	82	-	2n+6	22	12
Algorithm 1 (1/8 Ax FA)	1.4509k + 3.8435(n-k) + 0.834	29.1894	8%	4k+10(n-k)+3	77	6%	2n+6	22	12
Algorithm 1 (5/8 Ax FA)	1.4509k + 3.8435(n-k) + 0.834	19.6190	38%	4k+10(n-k)+3	53	35%	2n+6	22	12
Algorithm 2 (1/8 Ax FA)	1.6694k + 3.8435(n-k) + 0.865	29.4389	7%	5k+10(n-k)+3	78	5%	2n+6	22	12
Algorithm 2 (5/8 Ax FA)	1.6694k + 3.8435(n-k) + 0.865	20.7425	34%	5k+10(n-k)+3	58	29%	2n+6	22	12
Algorithm 3 [16] (1/8 Ax FA)	1.6678k + 3.8435(n-k) + 0.865	29.4373	7%	5k+10(n-k)+3	78	5%	2n+6	22	12
Algorithm 3 [16] (5/8 Ax FA)	1.6678k + 3.8435(n-k) + 0.865	20.7345	34%	5k+10(n-k)+3	58	29%	2n+6	22	12
Algorithm 4 (1/8 Ax FA)	1.8697k + 3.8435(n-k) + 0.865	29.6392	6%	5k+10(n-k)+3	78	5%	2n+6	22	12
Algorithm 4 (5/8 Ax FA)	1.8697k + 3.8435(n-k) + 0.865	21.7440	31%	5k+10(n-k)+3	58	29%	2n+6	22	12

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

in [16]. The resulting energy consumption per bit was 3.8435 nJ with an additional 0.8053 nJ for the extra steps. We recreated the serial [7], [19], and semi-parallel [10] adders for a fair comparison. The results can be seen in Table IX. The improvements of the parameter P if the presented algorithms to the others were determined via (16) and all the results were inserted into Table IX

$$\text{Improvement} = \frac{P_{\text{worse}} - P_{\text{better}}}{P_{\text{worse}}} \times 100\%. \tag{16}$$

It can be seen that the first algorithm since it only requires one work memristor, has a significantly lower energy consumption than the other algorithms. The second and third [16] algorithms have almost the same energy consumption, differing only by 1 pJ. The fourth algorithm performs significantly worse than the others due to the place of its approximation. Compared to the exact full adder in the semi-serial structure [8], a significant improvement of $6\% - 38\%$ can be seen for all the algorithms.

2) Number of Steps: The second important metric at the circuit-level is the number of steps (or the clock cycles) that are necessary per bit, since it represents the delay of the calculation. The first algorithm we presented requires four steps per bit and an additional step at the beginning of the calculation, which ensures that the work memristors are properly initialized, i.e., set to logical 0. Our other two algorithms and the approach from [16] need five steps for one bit and again an extra step to reset (initialize) the work memristors beforehand.

The exact algorithm in the semi-serial structure from [8] requires ten steps per bit and two extra steps which are applied only once per computation cycle. With a higher bit-width, the extra step of the presented algorithms loses strongly in importance. In comparison to an RCA with only exact adders, $5\% - 35\%$ fewer steps are required. Even compared to the parallel structure [9] which also requires five steps per bit, every algorithm presented in this article and [16] is faster since the parallel structure requires 16 extra steps. This is a noticeable difference for the RCA with few bits. A comparison of the required steps can be found in Table IX. We used an RCA with approximation degrees of 1/8 and 5/8. As the same trend applies to the approximation degrees in between, they were not shown in the table. The exact full adders that we used for the higher bits are taken from [8].

For n-bit adders, we calculated the number of steps for the first algorithm using (17), where the approximation degree is determined by the factor k, which represents the number of approximated full adders. The other three algorithms follow:

$$\text{Steps}(n, k) = 4k + 10(n - k) + 3 \tag{17}$$

$$\text{Steps}(n, k) = 5k + 10(n - k) + 3. \tag{18}$$

The improvement of all the algorithms was related to the semi-serial algorithm from [8] as the baseline, since it is the exact version of the proposed algorithms, and evaluated at 8-bit. For this the formula, (16) was used. The results of this can be seen in Table IX.

3) Area Usage: Another important comparison point at the circuit-level is the area usage, which represents the cost of the circuit. This is assessed by the number of required memristors and switches. The number of memristors required by the exact full adder in an RCA is always considered here. The exact and approximated full adder from [7], [19], and [30] all require $2n + 3$ memristors and no additional switches. The exact algorithm in the semi-serial topology from [8] and the algorithms Seiler and TaheriNejad [16] presented use $2n + 6$ memristors and 12 switches. As both the serial and semi-serial topologies scale with $2n$ they are approximately equal when many bits are used. As the parallel structure from [9] uses $4n+1$ memristors and n switches for the n-bit, the algorithms we presented are much more efficient area wise and require up to 50% less memristors. The comparison of the different algorithms' area usage can be found in Table IX.

B. Comparison to Approximate Full Adders

To give a comparison to the other approximated full adders that utilize IMPLY we compared the results of the evaluation for the algorithms from [29] and [30] with the algorithms presented in this work. We did not compare to the MRL-based approximated full adder from [42] and [43] and other approximated adders because the disparity to IMPLY-based structures is too significant to make a meaningful comparison. The overview of all relevant comparison points at circuit-level is presented in Table X, where we related our algorithms and the algorithm from [16] to the SIAFA 1, 3. We did not directly compare to [16] since the results are very similar to the second algorithm (due to their symmetry as explained in Section III-D) and as we wanted to compare the methodological approach as a whole with other adders. All comparisons were made for all algorithms with an approximation degree of 5/8.

1) Energy Consumption: In comparison to [29] and [30], the adders presented in this work are more energy efficient than any SIAFA or SAFAN adder. When compared to SIAFAs 1 and 3, the adders require $5\% - 17\%$ less energy, which increases to up to 29% when we compare our first algorithm to SIAFA 4. This is due to the better energy efficiency of the semi-serial topology.

2) Number of Steps: With the ability to perform some steps in parallel as explained in Section III, our algorithms require $43\% - 50\%$ fewer steps for an 8-bit calculation, which is a significant improvement, considering that both are approximated algorithms.

3) Area Usage: All the approaches have a similar area usage since they are in the order of $2n$ memristors for the n-bit adders. The adder in the semi-serial topology requires three more memristors and 12 additional CMOS switches.

4) Error Metrics: Since, the approximated adders from [30] and the adders presented by us and [16] share similar truth tables we expected resembling error metrics. Our fourth algorithm and SIAFA4 should produce the same results for MED, NMED, and MRED since they share the same truth table. This is true for the 8-bit simulation but not for the 16- and 32-bit cases. This deviation happens because we simulated these cases with only one million random input combinations. We explained this in more detail in Section IV. In the 8-bit case, our second and the third [16] algorithms differ less than 1% in NMED from SIAFAs 1 and 2 and exhibit a noticeably improved MRED of 2% percent. The first algorithm we presented performed worse than the algorithms above in both NMED and MRED but is more accurate than SIAFA 2 by 27% and 28% in NMED and MRED, which overall performs worst in terms of accuracy. In the simulations with more bits the relation of the algorithms in terms of precision stays about even. The most significant advantage of the algorithms presented in this work is that they excel in speed and energy efficiency while the area usage is only slightly higher than the algorithms from [29] and [30] for few bits and negligible for higher bits since both scale equally.

VI. Application in Image Processing

Image processing is a widely employed technology with diverse applications across various domains, including medicine, industry, automation, robotics, and media [47]. Given the elevated computational complexity inherent in these applications, adopting an approximate approach holds significant potential for substantial gains in both the energy efficiency and computational step reduction. Given the inherent error-resistant nature of these applications, they represent ideal candidates for identifying efficient tradeoffs.

We simulated the presented approximated adders in an RCA structure via MATLAB for different approximation degrees.

TABLE X
CIRCUIT-LEVEL COMPARISON TO APPROXIMATE FULL ADDER

Ax full adder	Energy consumption (nJ)		Improvement in comparison to SIAFA1,3	No. of steps		Improvement in comparison to SIAFA1,3	No. of memristors		No. of switches
	n, k	n=8-bit, k=5	n=8-bit, k=5	n, k	n=8-bit, k=5	n=8-bit, k=5	n, k	n=8-bit, k=5	n, k
SIAFA 1,3 [30]*	1.7090k + 4.8250(n-k)	23.0200	-	8k+22(n-k)	106	-	2n+3	19	0
SIAFA 2 [30]*	2.5131k + 4.8250(n-k)	27.0405	-15%	10k+22(n-k)	116	-9%	2n+3	19	0
SIAFA 4 [30]*	1.6628k + 4.8250(n-k)	23.0080	0%	8k+22(n-k)	106	0%	2n+3	19	0
SAFAN [29]*	1.7066k + 4.8250(n-k)	22.7890	1%	7k+22(n-k)	101	5%	2n+3	19	0
Algorithm 1	1.4509k + 3.8435(n-k) + 0.834	19.1690	17%	4k+10(n-k)+3	53	50%	2n+6	22	12
Algorithm 2	1.6694k + 3.8435(n-k) + 0.865	20.7425	10%	5k+10(n-k)+3	58	45%	2n+6	22	12
Algorithm 3 [16]	1.6678k + 3.8435(n-k) + 0.865	20.7345	10%	5k+10(n-k)+3	58	45%	2n+6	22	12
Algorithm 4	1.8697k + 3.8435(n-k) + 0.865	21.7440	6%	5k+10(n-k)+3	58	45%	2n+6	22	12

* We have simulated the circuits from [29], [30] similar to ours and obtained numbers that do not match theirs and we are not sure why. So for a fair comparison, we are reporting our own simulated results.

We assessed the degradation in accuracy on the application level using quality metrics, such as PSNR, SSIM, and MSSIM. We evaluated and analyzed the RCA in several specific applications, such as image addition, image subtraction, and gray-scale filtering, and determined their quality metrics, respectively. This analysis aimed to not only capture the error metrics outlined in Section IV but also to delve into the application-level behavior of each algorithm and find the boundaries of applicability for the proposed algorithms. Every algorithm presented by us was able to reach the 30 dB threshold in PSNR for every application with up to five out of eight adders being approximated.

A. Image Addition

Image addition stands as a fundamental application within image processing, commonly employed for the tasks, such as masking and enhancement through averaging [34]. Image addition entails the summation of corresponding pixels from the two images of identical dimensions, followed by halving the resultant values. As an example, we simulated two well-known 256 × 256 8-bit example images with all full adders presented in this work. We chose exactly these images so that we would have a direct comparison to the approximated adders from [16], [29], and [30]. We varied the approximation degree from one up to five approximated adders out of eight total adders. We found that the PSNR value surpasses the required threshold for all the algorithms with these approximation degrees. For the scenario where the quantity of approximated adders equals or exceeds 6, a PSNR value below 30 dB is observed across all the algorithms. This falls below the widely accepted threshold, indicating a discernible distortion in image quality. The simulated images for all of our algorithms and Algorithm 3 [16] with an approximation degree of 5/8 are shown in Fig. 5 and the calculated quality metrics for the image addition are presented in Table XI. With five approximated full-adders, the second and third [16] algorithms together with SIAFA 1 exhibit the best PSNR.

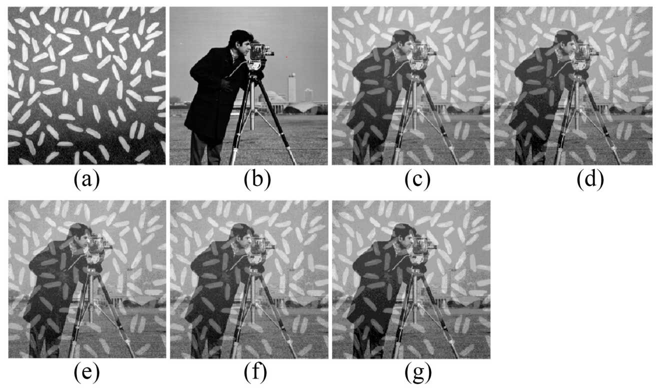


Fig. 5. Results of the RCA with of approximation degree of 5/8. (a) Rice. (b) Cameraman. (c) Exact Image Addition. (d) Algorithm 1. (e) Algorithm 2. (f) Algorithm 3 [16]. (g) Algorithm 4.

TABLE XI
QUALITY METRICS OF IMAGE PROCESSING

Algorithm	Image addition		Image subtraction		Gray-scale filter		Average performance	
	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM	PSNR (dB)	MSSIM
approximation degree: 1/8 Ax. full adder								
Algorithm 1	51.12	0.9976	55.81	0.9966	57.44	0.9993	54.79	0.9978
Algorithm 2	51.12	0.9976	58.76	0.9974	53.21	0.9985	54.36	0.9978
Algorithm 3*	51.12	0.9976	58.76	0.9974	52.90	0.9984	54.26	0.9978
Algorithm 4	51.12	0.9976	58.76	0.9974	54.16	0.9984	54.68	0.9978
approximation degree: 2/8 Ax. full adder								
Algorithm 1	48.46	0.9957	50.02	0.9869	51.37	0.9972	49.95	0.9933
Algorithm 2	47.16	0.9940	51.83	0.9911	49.44	0.9968	49.48	0.9940
Algorithm 3*	47.17	0.9941	51.80	0.9909	49.93	0.9970	49.63	0.9940
Algorithm 4	48.17	0.9952	52.16	0.9909	49.06	0.9955	49.80	0.9939
approximation degree: 3/8 Ax. full adder								
Algorithm 1	43.79	0.9884	45.56	0.9703	44.76	0.9878	44.70	0.9822
Algorithm 2	43.33	0.9858	45.46	0.9559	44.93	0.9903	44.57	0.9773
Algorithm 3*	43.31	0.9860	45.26	0.9520	45.49	0.9910	44.69	0.9763
Algorithm 4	43.41	0.9865	46.71	0.9765	43.06	0.9841	44.39	0.9824
approximation degree: 4/8 Ax. full adder								
Algorithm 1	38.09	0.9619	40.38	0.9409	37.97	0.9474	38.81	0.9501
Algorithm 2	38.20	0.9583	38.86	0.8210	39.87	0.9677	38.98	0.9157
Algorithm 3*	38.31	0.9603	39.45	0.8602	40.23	0.9693	39.33	0.9299
Algorithm 4	37.68	0.9576	40.92	0.9463	36.96	0.9451	38.52	0.9497
approximation degree: 5/8 Ax. full adder								
Algorithm 1	32.06	0.8966	34.93	0.9104	30.39	0.8163	32.46	0.8744
Algorithm 2	32.98	0.8901	33.57	0.6896	34.29	0.8989	33.57	0.8262
Algorithm 3*	32.93	0.8934	33.74	0.7183	34.05	0.9003	33.57	0.8373
Algorithm 4	32.06	0.8920	35.13	0.9130	31.51	0.8525	32.90	0.8858

* The quality metrics for algorithm 3 were taken from [16]

B. Image Subtraction

Image subtraction is often used for motion detection. But it is also used in robotics, medicine, or surveillance systems [48], [49]. The image subtraction procedure is very similar to the image addition. In this case, we are representing the pixels of two images of the same size as 2 s complement. We then take the inversion for each pixel of the subtracted image. After this every corresponding pixel, of the first and the inverted second image, is added together in our RCA structure. As an example, we took two 512 × 512 8-bit images from the image database of [50] and simulated the subtraction in MATLAB. We again choose these images to have an apple-to-apple comparison with the adders from [16], [29], and [30]. The results of the different algorithms with an approximation degree of 5/8 can be seen in Fig. 6. The simulated quality

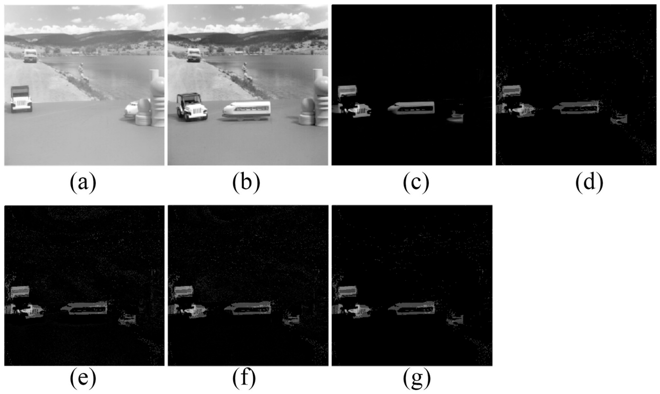


Fig. 6. Results of the RCA with an approximation degree of 5/8. (a) First image [50]. (b) Second image [50]. (c) Exact image subtraction. (d) Algorithm 1. (e) Algorithm 2. (f) Algorithm 3 [16]. (g) Algorithm 4.

metrics for the image subtraction can be found in Table XI. Again the PSNR value of all the algorithms is over 30 dB for an approximation degree of up to 5/8. With six or more adders this threshold again could not be reached and the noise effects would render motion detection applications unusable. For this application Algorithm 4 and SIAFA 4 exhibit the best PSNR and MSSIM, closely followed by Algorithm 1.

C. Gray-Scale Filter

The gray-scale filter converts a colored RGB image into a gray-scale version. In an image, each pixel comprises three colors: 1) red; 2) green; and 3) blue along with their corresponding intensities. To produce a gray-scale image, the algorithm sums up the individual color values for each pixel and then divides the resulting sum by three. With this, the resulting gray-scale intensity is the average of all the three color values. We first added the red and the green color-space together and after that added the red to the prior result. It is noteworthy that in alternative gray-scale conversion methods, the color components are not uniformly weighted, leading to disparate outcomes in the generated gray-scale images. We performed the mentioned process for all the pixels of the 684 $\times$ 912 8-bit example image, which was again chosen so that a comparison to the SoA adders could be drawn fairly. We simulated every proposed algorithm with different approximation degrees up to five out of eight approximated adders. An overview of the quality metrics we assessed is located in Table XI. Each algorithm exhibits more than 30 dB PSNR at approximation degrees of 1–5 and is visually almost indistinguishable from the exact calculation. The simulation result of all the algorithms with an approximation degree of 5/8 can be found in Fig. 7. The results of six or more approximated adders did not meet the required PSNR threshold of 30 dB. This time SIAFAs 1 and 3 exhibit the best PSNR, followed by Algorithms 2 and 3. All the four approaches share roughly the same MSSIM.

D. Application-Level Comparison

1) With Exact Semi-Serial Adder [8]: To effectively compare our algorithm with the exact approach [8] in image processing we looked at the difference per pixel for each application. We compared the RCA structures with five out of eight approximated adders with the exact 8-bit RCA, which is the highest sufficient approximation degree. In image addition and subtraction only one addition is required per pixel, while in the gray-scale filter, two additions are required. If we sum up the required energy and steps per pixel, we get the total energy consumption and number of steps per image processing application. With the algorithms 1, 2, and 4 up to 38%, 34%, and 31% less energy and 35%, 29%, and 29% fewer steps are required for the image processing applications. Algorithm 3 from [16] has almost identical (< 0.1% difference) results as Algorithm 2, which again can be explained by their symmetry to each other. For these gains in speed and energy efficiency, the accuracy of the calculations is reduced but still in an acceptable range as shown in Table XI. As the gray-scale filter requires two additions the improvement is correspondingly higher than with the other image processing applications. With the presented 684 $\times$ 912 8-bit image we were able to reduce the number of steps (the clock cycles) by about 36 million and the required energy by 14.9 mJ in comparison to the exact calculations. We achieve higher gains than Algorithm 3 from [16] by six million steps and 1.4 mJ, a significant improvement over an already efficient adder.

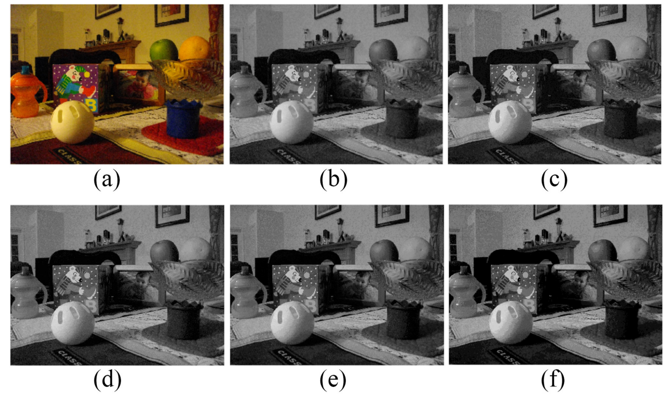


Fig. 7. Results of the RCA with an approximation degree of 5/8. (a) Toysnoflash. (b) Exact gray-scale filter. (c) Algorithm 1. (d) Algorithm 2. (e) Algorithm 3 [16]. (f) Algorithm 4.

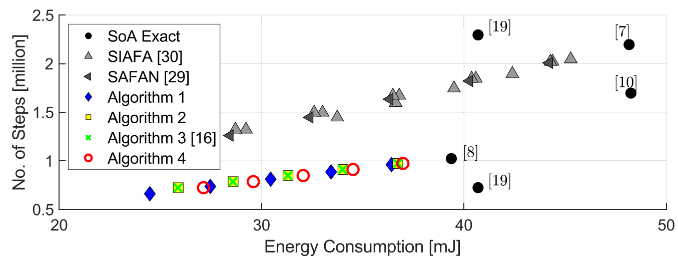


Fig. 8. Application-level comparison with 1/8 to 5/8 approximated adders for the gray-scale filter example.

2) With Approximated Adder From [29] and [30]: Since, the topology from [29] and [30] differs from the semi-serial structure used here and in [16], stark differences in energy consumption and number of steps are expected. In all the image processing applications, the adders presented in this work require 5% – 29% less energy and 43% – 50% fewer steps. The results for our experiments with 5/8 approximated adders can be seen in Table XII, where our approach saves up

TABLE XII
APPLICATION LEVEL COMPARISON TO EXACT SEMI-SERIAL ADDER [8], AND APPROXIMATE SERIAL ADDERS [29], [30]

Algorithm	Image addition (256x256 8-bit Image)				Image subtraction (512x512 8-bit Image)				Grayscale filter (684x912 8-bit Image)			
	Energy per pixel (nJ)	Total Energy (mJ)	Steps per pixel	Total Steps (million)	Energy per pixel (nJ)	Total Energy (mJ)	Steps per pixel	Total Steps (million)	Energy per pixel (nJ)	Total Energy (mJ)	Steps per pixel	Total Steps (million)
Semi-Serial Exact [8]	31.558	2.068	82	5.374	31.558	8.273	82	21.496	63.116	39.372	164	102.305
Algorithm 1 (5/8 Ax FA)	19.619	1.286	53	3.473	19.619	5.143	53	13.893	39.238	24.477	106	66.124
Algorithm 2 (5/8 Ax FA)	20.743	1.359	58	3.801	20.743	5.438	58	15.204	41.486	25.879	116	72.362
Algorithm 3 [16] (5/8 Ax FA)	20.735	1.305	58	3.801	20.735	5.436	58	15.204	41.470	25.869	116	72.362
Algorithm 4 (5/8 Ax FA)	21.744	1.425	58	3.801	21.744	5.700	58	15.204	43.488	27.128	116	72.362
SIAFA 1,3 [30] (5/8 Ax FA)	23.020	1.509	106	6.947	23.020	6.035	106	27.787	46.040	27.209	212	125.287
SIAFA 2 [30] (5/8 Ax FA)	27.041	1.772	116	7.602	27.041	7.089	116	30.409	54.082	31.961	232	137.106
SIAFA 4 [30] (5/8 Ax FA)	23.008	1.508	106	6.947	23.008	6.031	106	27.787	46.016	27.194	212	125.287
SAFAN [29] (5/8 Ax FA)	22.789	1.493	101	6.619	22.789	5.974	101	26.477	45.578	26.936	202	119.377

to 7.5 mJ and 71 million steps compared to the SoA approximations. We plotted the energy-speed (number of steps) of our approaches and the SoA algorithms for the gray-scale filter example with different approximation degrees in Fig. 8. We can see the efficiency of our algorithms, while reaching equal image quality in most cases and comparable quality in others. The following comparisons were made with an approximation degree of 5/8 since it is the highest approximation degree with acceptable image quality. In the image addition, the second and third [16] algorithms exhibit almost equal PSNR and MSSIM as the best algorithms from [30]. The other two algorithms from this work have about 0.9 dB less PSNR but display similar MSSIM. The SAFAN adder from [29] yields the worst results with a PSNR of only 30.59 dB. In the image subtraction, all of the presented algorithms have better PSNR in comparison to SIAFAs 1, 2, and 3 by at least 1 dB. The fourth algorithm, SIAFA4 and SAFAN perform the best with over 35 dB PSNR. SIAFAs 2 and 4, and our first and fourth algorithms exhibit the best MSSIM with over 0.9 in this application. At the gray-scale filter, SIAFAs 1 and 3 have better PSNR than our presented algorithms and Algorithm 3 [16]. The second algorithm still shows very good results with a PSNR over 34 dB. On average the second and third [16] algorithms together with SIAFA 1 and 3 perform best in terms of PSNR, followed by our first and fourth algorithms as well as SIAFA 4. SIAFAs 2 and 4 and Algorithms 1 and 4 from this work show the best MSSIM on average.

VII. CONCLUSION

In this work, we presented three novel approximated full adders based on the memristive IMPLY logic in the semi-serial topology for the in-memory image processing. The primary emphasis was on reducing the necessary steps per computation while showcasing a commendable tradeoff between the area consumption, speed, energy consumption, and accuracy. By implementing the proposed methodology, we observed a reduction in energy consumption of $6\%-38\%$ when compared to the exact full adder in the semi-serial topology and $5\% - 29\%$ compared to the other approximated approaches. We were able to reduce the required number of steps by $5\% - 35\%$ compared to the exact adder and $43\% - 50\%$ to the other approximated adders at the same approximation degree. We demonstrated the fastest IMPLY-based adder algorithm, which requires a mere 53 steps for an 8-bit computation and is even faster than the algorithm in the parallel structure (requiring 56 steps). We integrated the approximated full adders as the lower bits in an RCA, simulated their behavior, verified their functionality, and assessed the error metrics. We applied the presented algorithms in various image processing applications, such as image addition, image subtraction, and gray-scale filtering. We evaluated the performance of the proposed image processing systems using varying approximation degrees and determined the quality of the resulting image with quality metrics. Our results indicate that for up to 5 bits of approximated adders in an 8-bit RCA, the image quality is deemed sufficient since the PSNR was over 30 dB. We can also see that different approximations excel in different applications, indicating that the error placement is crucial and highly application specific. We showed how this approach leads to drastic improvements in speed and energy consumption at the application level. An in-depth stochastic analysis of the proposed algorithms, their application for the 16- and 32-bit systems, and a more generalized theory about the effects of approximations on the image processing are domains for the future research.

VIII. ACKNOWLEDGMENT

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

REFERENCES

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

[2] R. S. Williams, *Finding the Missing Memristor*, Stanford Univ., Stanford, CA, USA, 2010.

[3] J. Borghetti et al. “Memristive switches enable stateful logic operations via material implication,” *Nature*, vol. 464, no. 4, pp. 873–876, Apr. 2010.

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

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

[6] D. Radakovits and N. Taherinejad, “Behavioral leakage and intercycle variability emulator model for rerams (BELIEVER),” 2021, *arXiv:2103.04179*.

[7] 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. (CCECE)*, 2017, pp. 1–4.

[8] N. TaheriNejad et al., “A semi-serial topology for compact and fast imply-based memristive full adders,” in *Proc. 17th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, 2019, pp. 1–4.

[9] 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.

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

[11] 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.

[12] 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*, 2011, pp. 409–414.

[13] 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)*, 2018, pp. 845–848.

[14] H. Jiang et al., "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 1–34, 2017.

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

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

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

[18] D. B. Strukov et al., "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[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, 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. dissertation, lŠécole Nationale Superieure Mines-Telecom Atlantique, Bretagne Pays de la Loire-IMT Atlantique, Brest, France, 2020.

[22] D. Radakovits et al., "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.

[23] S. E. Fatemieh et al., "Approximate in-memory computing using memristive IMPLY logic and its application to image processing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2022, pp. 3115–3119.

[24] 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)*, 2011, pp. 142–147.

[25] 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)*, 2018, pp. 1–7.

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

[27] 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, 2016.

[28] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL—Memristor ratioed logic," in *Proc. 13th Int. Workshop Cell. Nanoscale Netw. Appl.*, 2012, pp. 1–6.

[29] 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. Elect. Eng.*, vol. 113, Jan.'2024, Art. no. 109053.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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. Exhib. (DATE)*, 2016, pp. 660–665.

[35] 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)*, 2020, pp. 1–5.

[36] Z. Yang, J. Han, and F. Lombardi, "Transmission gate-based approximate adders for inexact computing," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, 2015, pp. 145–150.

[37] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–13, Mar. 2016.

[38] 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.

[39] Z. Wang et al., "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, pp. 600–612, 2004.

[40] G.-H. Chen, C.-L. Yang, L.-M. Po, and S.-L. Xie, "Edge-based structural similarity for image quality assessment," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 2, 2006, pp. 1–4.

[41] 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, 2017, Art. no. 1750082.

[42] S. Muthulakshmi, C. S. Dash, and S. R. S. Prabaharan, "Memristor augmented approximate adders and subtractors for image processing applications: An approach," *AEU Int. J. Electron. Commun.*, vol. 91, no. 5, pp. 91–102, 2018.

[43] S. Muthulakshmi, C. S. Dash, and S. R. S. Prabaharan, *Memristor-Based Approximate Adders for Error Resilient Applications*. Singapore: Springer, 2018, pp. 51–59.

[44] K. Bickerstaff and E. E. Swartzlander, "Memristor-based arithmetic," in *Proc. Conf. Record 44th Asilomar Conf. Signals, Syst. Comput.*, 2010, pp. 1173–1177.

[45] D. Radakovits et al. "Second (v2.0) LTSpice implementation of VTEAM." Sep. 2019. [Online]. Available: https://www.ict.tuwien.ac.at/staff/taherinejad/projects/memristor/files/vteam2.asc

[46] "Knowm SDC memristors." Feb. 2024. [Online]. Available: http://knowm.org/downloads/Knowm_Memristors.pdf

[47] M. Khaleqi, M. Ahmadinejad, and M. H. Moaiyeri, "Ultraefficient imprecise multipliers based on innovative 4:2 approximate compressors," *Int. J. Circuit Theory Appl.*, vol. 49, no. 9, pp. 169–184, 2020.

[48] R. B. Paranjape. "Fundamental enhancement techniques," in *Handbook of Medical Imaging*, I. N. Bankman, Ed., Cambridge, MA, USA: Academic, 2000, pp. 3–18.

[49] A. Fernández-Caballero, J. C. Castillo, J. Martínez-Cantos, R. Martínez-Tomás, "Optical flow or image subtraction in human detection from infrared camera on mobile robot," *Robot. Auton. Syst.*, vol. 58, no. 12, pp. 1273–1281, 2010.

[50] *Signal and Image Processing Institut (SIPI)*, Univ. South. California, Los Angeles, CA, USA, 2017.