Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/JXCDC.2023.3238030

Approximated 2–Bit Adders for Parallel In–Memristor Computing with a novel Sum–of–Product Architecture

CHRISTIAN SIMONIDES[1*], DOMINIK GAUSEPOHL[1*], PETER M. HINKEL[1*], FABIAN SEILER[2] and NIMA TAHERINEJAD[1,2]

[1]Institute of Computer Engineering, Heidelberg University, Heidelberg, Germany
[2]Institute of Computer Technology, Technische Universität Wien (TU Wien), Vienna, Austria

CORRESPONDING AUTHOR: CHRISTIAN SIMONIDES (e-mail: christian.simonides@stud.uni-heidelberg.de).

* These authors contributed equally to this work

ABSTRACT Conventional computing methods struggle with the exponentially increasing demand for computational power, caused by applications including image processing and Machine Learning (ML). Novel computing paradigms such as In–Memory Computing (IMC) and Approximate Computing (AxC) provide promising solutions to this problem. Due to their low energy consumption and inherent ability to store data in a non–volatile fashion, memristors are an increasingly popular choice in these fields. There is a wide range of logic forms compatible with memristive IMC, each offering different advantages. We present a novel mixed–logic solution that utilizes properties of the Sum–of–Products (SOP) representation and propose a full–adder circuit that works efficiently in 2–bit units. To further improve the speed, area usage, and energy consumption, we propose two additional Approximate (Ax) 2–bit adders that exhibit inherent parallelization capabilities. We apply the proposed adders in selected image processing applications, where our Ax approach reduces the energy consumption by $31\,\%$–$40\,\%$ and improves the speed by $50\,\%$. To demonstrate the potential gains of our approximations in more complex applications, we applied them in ML. Our experiments indicate that with up to $6/16$ Ax adders there is no accuracy degradation when applied in a Convolutional Neural Network (CNN) that is evaluated on MNIST. Our approach can save up to $125.6\,\mathrm{mJ}$ of energy and 505 million steps compared to our exact approach.

INDEX TERMS Adder, Approximate Computing, In–Memory Computing, Image Processing, Machine Learning, Memristor

I. INTRODUCTION

In recent years, demanding workloads like Machine Learning (ML) applications require an ever–increasing amount of computational capability. At the same time, traditional processors are limited by their power consumption [1], and their exponential energy demand is becoming a concern [2]. This motivates research in novel computing paradigms and emerging technologies such as memristors [2], [3], [4]. They are power efficient, have a small footprint, and store data in non–volatile fashion with their resistive states [4], [5], [6], [7]. Memristors are compatible with crossbar arrays and can compute operations directly in memory, making them the ideal targets for In–Memory Computing (IMC) [5], [8].Various stateful logic forms based on memristors have been presented [7], [8], [9], [10], [11], each optimized for specific applications.However, there are still a lot of untouched potential gains in combining the advantages of different approaches and utilizing them in a mixed logic. Another emerging approach is Approximate Computing (AxC), which can drastically improve energy consumption, speed, and area usage. The trade–off for these gains is a reduction in accuracy, meaning this paradigm is only applicable in error–resilient applications such as image processing, ML, or related fields such as computer vision [12], [13], [14]. In this work, we propose a highly flexible, fast, and energy–efficient mixed logic form for memristive IMC that is based on MAGIC [9] and FELIX [10]. We implemented an exact full adder based on the presented logic and further improved

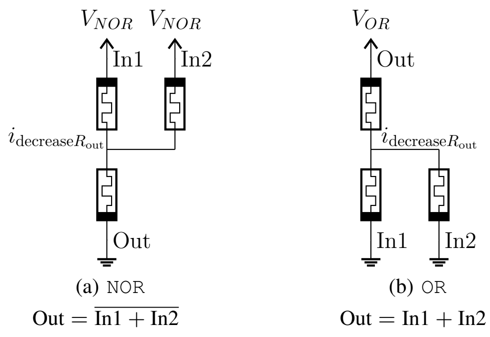

V_{NOR}
V_{NOR}
In1
In2
$i_{\text{decrease}R_{\text{out}}}$
Out
(a) NOR
Out $= \overline{\text{In1} + \text{In2}}$
V_{OR}
Out
$i_{\text{decrease}R_{\text{out}}}$
In1
In2
(b) OR
Out $= \text{In1} + \text{In2}$

FIGURE 1. **Memristive circuits for MAGIC [9] NOR and FELIX [10] OR.**

the efficiency by introducing two Approximate (Ax) versions of the adder circuit.Instead of the conventional Ripple Carry Adder (RCA) approach [15], our approximations are based on fully parallelizable 2–bit units. We applied our adders in image processing and ML, where we drastically improved speed and energy consumption. This work is divided into eight sections. In Section II we review the literature and cover related work. The proposed design methodology and Ax adders are described in Section III. In Section IV we simulate the presented exact and Ax adders, discuss different parameter variations for our approach, and evaluate the error metrics. In Section V we compare to State–of–the–Art (SoA) logic forms and approximations. We evaluate three image–processing applications and show the potential gains in Section VI. In Section VII we showcase the applicability of our approximations in ML and conclude the paper in Section VIII.

II. RELATED WORK

A. MEMRISTOR–BASED IN–MEMORY–COMPUTING (IMC)

Memristors are two–terminal components that store data in a non–volatile fashion using their resistive states [3]. With advantages such as low power consumption, fast write time, and small dimensions, they are ideal as memory cells [3], [5], [6], [16]. The memristor's minimum (R_{on}) and maximum (R_{off}) resistive values are conventionally interpreted as logical '1' and '0', respectively, which can be reached by an applied voltage. Various stateful logic forms for in–memristor computing, such as Memrisor–Aided Logic (MAGIC) [9], Fast and Energy–Efficient Logic (FELIX) [10], Memristor–Based Material Implication (IMPLY) [8], Three Memristors Stateful Logic (TMSL) [11], and Single–Cycle In–Memristor XOR (SIXOR) [7] have been proposed. MAGIC and FELIX stand out due their low power consumption, non–destructive operations, and simple architecture [9], [10]. In crossbar–arrays MAGIC NOR and FELIX OR provide an efficient and flexible atomic logic set. They are shown in Fig. 1 and require only a single voltage pulse, compared to other logic forms that have more complex requirements [7], [8], [11].

B. APPROXIMATE COMPUTING (AxC)

With AxC performance metrics such as energy efficiency, speed, and area usage can be significantly improved, whereas the accuracy is decreased as a trade–off. Error metrics were introduced and applied in the SoA to evaluate the degree of inaccuracy [13], [17]. For more detailed information on the error metrics, we refer the reader to [13], [17], [18]. The applications of erroneous calculations are limited to applications that exhibit inherent error–resilience, such as image processing and ML as well as related fields [12], [13], [14]. A commonly used quality metric for image processing is the Peak Signal to Noise Ratio (PSNR), which indicates the noise level. A value of over 30 dB is considered acceptable in the literature [13], [17], [18]. The Structural Similarity Index Measure (SSIM) is also often used to quantify the perceived image similarity [19]. Recently, lots of interest was shown toward memristor–based AxC. In [20] the authors automatically evaluated error metrics of Ax adders but provided no specific implementations to compare. Ax adders based on IMPLY were presented in [15], [21], [22], [23], [24] and evaluated in various image processing applications. The algorithms from [15] stand, out as they presented the fastest and energy–efficient IMPLY–based algorithms in many topologies, which we will use as a baseline for comparison. Here, we present a mixed logic that combines the advantages of MAGIC and FELIX to provide an efficient and flexible design methodology for arbitrary logic functions. We showcase the potential of our approach by implementing an exact adder and two approximated derivatives that are based on fully parallelizable 2–bit units. When applied to image processing and ML, our method leads to significant improvements in energy, area, and latency.

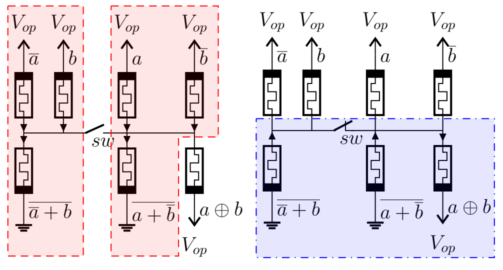
III. PROPOSED APPROACH

A. DESIGN METHODOLOGY

The core concept of our work is to combine the MAGIC [9] NOR (or FELIX NOR as they are similar) with the FELIX [10] OR circuits. With this, we can evaluate arbitrary boolean expressions with up to five inputs in only three cycles. An overview of the procedure is given in Algorithm 1. To achieve this, a boolean equation f must be converted to the Sum–of–Products (SOP) form:

$$f(x_1, x_2, \ldots, x_n) = \sum_{m_i \in M} \left(\prod_{j=1}^{n} x_j^{(m_i)} \right), x_j^{m_i} \in \{1, x_j, \overline{x_j}\}, \tag{1}$$

where f is expressed as a sum of the minterms m_i for n inputs. The expression can be converted to only OR and NOR statements via De Morgan's theorem $\left(\overline{A} \cdot \overline{B} \equiv \overline{A + B}\right)$. The proposed approach consists of three steps. In the first, the values in the minterms m_k^p are populated, which are then used in the second step to calculate MAGIC NOR (m_k^p). The resulting intermediate results are then combined with FELIX OR. With our method, multiple output functions can be computed in parallel as the created minterms are independent of each other, enabling stark parallelization. We require both the inputs and inversions of inputs to create valid minterms m_k^p for the output p. For that, we use the MAGIC NOR gate with a single input, which is practically a NOT gate. We do



(a) Phase I (NOR) (b) Phase II (OR)

FIGURE 2. Circuit design for XOR using the presented logic. In phase I, the intermediate results $\overline{\overline{a}+b}$ and $\overline{a+\overline{b}}$ are calculated with NOR (red markings). In phase II, the intermediate results are combined with OR (blue markings), storing $a \oplus b$ in the output.

this operation concurrently for all of our function inputs to get the corresponding inversions. Because these (inverted) inputs may be required by multiple minterms m_k^p, their values must be distributed. As we are working in a crossbar array, the inputs of all the required minterms can be SET or RESET in one cycle and in parallel by the control logic. The memristors that are used for the intermediate results s_k^p must also be initialized to '1' and the output memristors o^p must be set to '0' [9], [10]. After the initialization, the inputs m_k^p are combined with a NOR function to calculate all intermediate results s_k^p, which is done parallel for all k modules. To achieve this, the operation voltage V_{op} is applied to all input memristors for each minterm, while s_k^p are grounded. To calculate the outputs, additional switches sw_k are required to connect the memristors holding the values s_k^p. These switches are not part of the crossbar array itself and are rather located in the control logic. When these switches sw_k are closed, the intermediate values s_k^p are connected to a common bus. We then apply OR with the intermediate values s_k^p as the inputs, by applying v_{op} to the output memristor. An advantage of our approach is that only a single operational voltage V_{op} is required, which we discuss in more detail in Section IV-B.

In Fig. 2, we illustrated an exemplary circuit for an XOR operation, which is represented as $\overline{\overline{\overline{a}+b}+\overline{a+\overline{b}}}$ within our approach. We marked the circuits for the NOR phase in red, where the intermediate results are stored in the memristors at the bottom. The OR phase is shown in blue, where the intermediate results are combined via an OR operation and stored in the output memristor. During both steps, only the memristors inside of the highlighted region(s) are connected to the bus. The switch sw is closed in the second phase.

B. FULL ADDER IMPLEMENTATION

As adders are the most basic building blocks, they are often used for comparison between different approaches. Therefore, we implemented an adder using the method proposed in this work. Since our approach is always calculated in two steps, we propose 2–bit adders instead of the conventional 1–bit, which are then combined into an RCA. This results in a discrete circuit with five inputs and three outputs $(s_i, s_{i+1}$ and $c_{out})$, which is still computed in two steps.

Algorithm 1: SOP function with q outputs

```
  Input: $\mathcal{I} = \{i_j\}_{j=0}^{n-1}$; Output: $\mathcal{O} = \{o^p\}_{p=0}^{q-1}$
1 procedure SOP Function $f : \mathcal{I} \mapsto \{0,1\}^q$
2     Create $\mathcal{I}^{not} = \{\mathtt{NOT}(i_j) \mid i_j \in \mathcal{I}\}$
3     for every output $o^p$ do in parallel:
4         Set $m_k^p = \{a_l \mid a_l \in \mathcal{I} \cup \mathcal{I}^{not}\}_{l=0}^{n-1}$
5         $sw_k$ open; $s_k^p \leftarrow \mathtt{NOR}(m_k^p)$, $\mathcal{S}^p = \{s_k^p\}$
6         $sw_k$ closed; $o^p \leftarrow \mathtt{OR}(\mathcal{S}^p)$
7     end
8 end
```

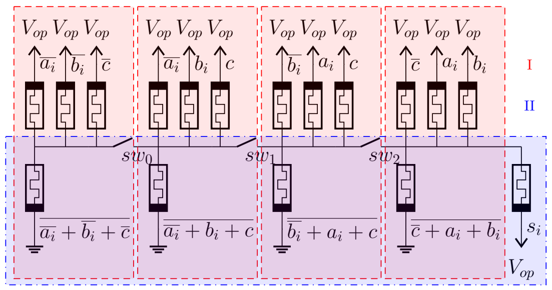

V_{op}
$\overline{a_i}$
$\overline{b_i}$
$\overline{c}$
b_i
c
a_i
I
II
sw_0
sw_1
sw_2
$\overline{\overline{a_i}+\overline{b_i}+\overline{c}}$
$\overline{\overline{a_i}+b_i+c}$
$\overline{\overline{b_i}+a_i+c}$
$\overline{\overline{c}+a_i+b_i}$
s_i

FIGURE 3. Proposed circuit for the output (s_i) of the exact 2–bit full adder. We marked the NOR circuits in red and the OR circuit in blue.

When we apply our method to the full adder logic of the 2–bit cells, three independent equations are evaluated concurrently using the proposed NOR and OR phases. We presented the circuit for s_i in Fig. 3, where the two phases are again color–coded. The other two circuits follow the same principle.

C. PROPOSED APPROXIMATED ADDER

To further improve the efficiency of memristive adders, an upcoming approach is AxC, where the accuracy is traded for improved speed, area usage, and energy efficiency [15], [21]. We analyzed our exact 2–bit adder to find approximations that are independent of c_{in}, so they can be computed in parallel. When the c_{in} of the 2–bit units is assumed as '0', and therefore disregarded, and we approximated the internal carry as b_i, we achieve the following equations:

$$s_i = a_i\overline{b_i} + \overline{a_i}b_i \tag{2}$$

$$\begin{aligned} s_{i+1} = &\ a_{i+1}b_{i+1}b_i + a_{i+1}\overline{b_{i+1}}\ \overline{b_i} + \\ &\ \overline{a_{i+1}}\overline{b_{i+1}}b_i + \overline{a_{i+1}}b_{i+1}\overline{b_i} \end{aligned} \tag{3}$$

$$c_{out} = a_{i+1}b_{i+1} + a_{i+1}b_i + b_{i+1}b_i \tag{4}$$

This approximation of s_i, s_{i+1}, and c_{out} prevents dependencies between the two additions $a_i + b_i$ and $a_{i+1} + b_{i+1}$ while making the three outputs independent of c_{in}. Since the Ax 2–bit units disregard c_{in}, the c_{out} only has to be computed on the last erroneous adder unit, allowing us to omit the circuit responsible for c_{out}. This leaves us with two possible implementations, where the c_{out} is either calculated on the last Ax bit or completely disregarded in every unit. We name our approximations: **P**arallel **2**–bit **A**pproximated **A**dder (with **C**arry–out), resulting in P2AAC and P2AA for both versions. The truth tables for both versions are shown in

TABLE I. Truth table of the proposed P2AAC circuit with the outputs "$\mathtt{c_{out}s_1s_0}$". The incorrect by-design outputs are marked in red.

a_1b_1	Input Bits $a_0b_0c_{in}$							
	000	001	010	011	100	101	110	111
00	000	000	011	011	001	001	010	010
01	010	010	101	101	011	011	100	100
10	010	010	101	101	011	011	100	100
11	100	100	111	111	101	101	110	110

TABLE II. Truth table of the proposed P2AA circuit ($\mathtt{c_{out}} = 0$) with the outputs "$\mathtt{c_{out}s_1s_0}$". The incorrect by-design outputs are marked in red.

a_1b_1	Input Bits $a_0b_0c_{in}$							
	000	001	010	011	100	101	110	111
00	000	000	011	011	001	001	010	010
01	010	010	001	001	011	011	000	000
10	010	010	001	001	011	011	000	000
11	000	000	011	011	001	001	010	010

Tables I and II, where the errors introduced by our approximation are marked in red. From Equations (2), (3) and (4) we can derive the necessary operations that an Ax 2–bit adder has to perform and design the circuit accordingly. Please note that the logic for s_i constitutes an XOR operation, which we show in Fig. 2. Due to the parallelization capabilities of our approximations, the number of steps required by a partially Ax RCA can be drastically reduced. Together with the initialization of the required inputs, our approach requires three steps for a 2–bit addition. The number of total steps is calculated by

$$S_{\text{P2AAC}}(k,n) = \tfrac{3}{2}\cdot(n-k)+3 \tag{5}$$

$$S_{\text{P2AA}}(k,n) = \max\left(3, \tfrac{3}{2}\cdot(n-k)\right), \tag{6}$$

where k corresponds to the number of Ax bits (must be multiples of two) and n is the total number of bits in the RCA. For every two bits, we require three steps, but depending on the Approximation Degree (AxD), some of those can run in parallel leading to fewer steps overall. We will denote the AxD of an RCA with an approximated part as k/n, where k is the number of Ax bits and n is the total number of bits of the RCA. It is noticeable that P2AAC requires three more steps than P2AA. This stems from the fact that with P2AA, no carry–out is produced, so the first two bits of the exact adder can also be computed in parallel to the Ax 2–bit units.

IV. CIRCUIT–LEVEL SIMULATION

A. SIMULATION SETUP

We verified the functionality of the proposed adders with LTSpice simulations. The SPICE implementation [25] of a memristor model based on Voltage–controlled ThrEshold Adaptive Memristor (VTEAM) [26] was used. The parameters for this model were set similarly to Table III [1], as

TABLE III. Parameters of the utilized VTEAM memristor model [25], [26].

Parameter	V_{off}	V_{on}	α_{off}	α_{on}	R_{off}	R_{on}
Value	$0.7\,\text{V}$	$-10\,\text{mV}$	3	3	$1\,\text{M}\Omega$	$10\,\text{k}\Omega$

k_{on}	k_{off}	w_{off}	w_{on}	w_{c}	a_{off}	a_{on}
$-0.5\,\text{nm/s}$	$1\,\text{cm/s}$	$0\,\text{nm}$	$3\,\text{nm}$	$107\,\text{pm}$	$3\,\text{nm}$	$0\,\text{nm}$

they are fitted to a real discrete Knowm memristor [28] that is commonly used [7], [15], [21], [29]. This increases our confidence in the practical relevance of our simulations and allows for an easier comparison. We note here that, similar to the difference between discrete and integrated Complementary Metal Oxide Semiconductor (CMOS) devices, discrete memristors have higher energy consumption and operate slower. It is reasonable to assume that integrated memristors provide a significant performance improvement [16], [30]. We assume that the circuit is mapped to a 1T1M crossbar array where the switches are located underneath the crossbar [31], while the external switches that connect the NOR modules are embedded in the control logic. We chose to follow the established practice from [21], [23], [29] on disregarding the effect of the switches and rather focusing on the logic design of the proposed circuits.

B. SIMULATION RESULTS

We simulated all possible input combinations of our proposed (Ax) adders and validated them with our theoretical calculations. All memristors were initialized accordingly. The inputs were set to their logical states, the memristors for the intermediate NOR step were set to logical '1', and the output memristors to '0'. We evaluate a grid of varying operation voltages (V_{op}) and cycle times (t_{op}), to find combinations that lead to valid results for arbitrary functions with up to five inputs. We implemented this limitation as the Fan-In of the OR step grows very fast for functions with more inputs. As these values strongly depend on the memristor model and technology nodes used, they should rather be seen as a case study to showcase the feasibility of our approach. The valid parameter combinations are shown in Fig. 4, where the energy efficiency and the quality of the logical output state are indicated as well. A value of over 0.66 is deemed logical '1', a value under 0.33 logical '0', and undefined otherwise. A valid combination is given if these limits are upheld for all input combinations. With quality, we denote the mean difference between optimal results ('0'/'1') and simulated outputs over all input combinations. Our case study revealed that only some combinations of V_{op} and t_{op} allow the correct operation of both NOR and OR gates with a varying number of inputs. The upper bound is given by a five input XOR, where the limiting factor is the high Fan-In to the OR gate (2^{n-1}), which are connected in parallel and can therefore push the total resistance below the required threshold. The lower bound of valid combinations is set when only a single input OR in a $NOT(i_0)$ arrangement, with a high input resistance, is used. With the cycle time, the total current flow through the output memristor can be limited,

[1]We used tungsten chalcogenide memristors produced by KNOWM [27]. Detailed information on the model and memristor can be found in [26], [28]

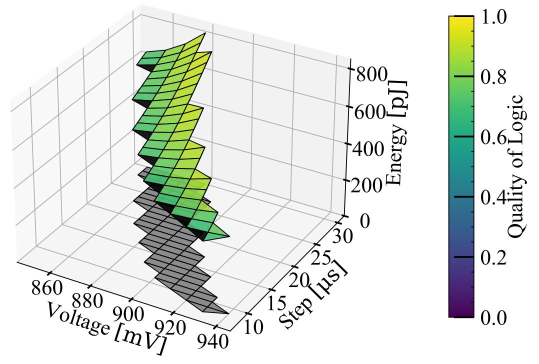


FIGURE 4. Resulting energy consumption and quality of logic outputs (deviation from ideal) for varying combinations of t_{op} and V_{op}.

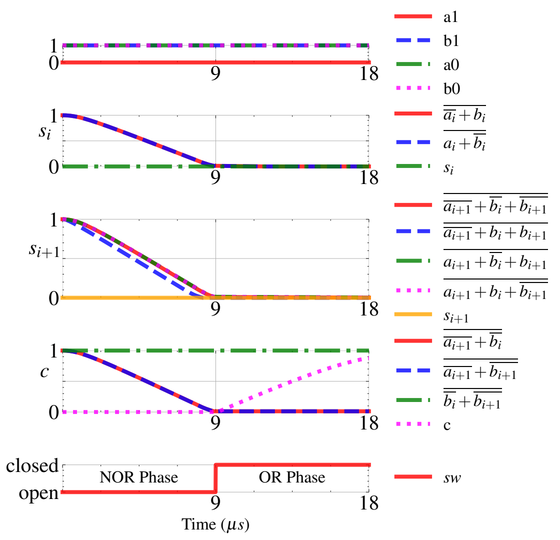


FIGURE 5. Example calculation of P2AAC with the inputs $a_{i+1,i}$: "01", $b_{i+1,i}$: "11". The intermediate results are calculated with NOR between $0-9\,\mu s$. The outputs s_i, s_{i+1}, and c are computed with OR from $9-18\,\mu s$. The three outputs are calculated in parallel.

which allows us to match it to the corresponding voltages. Our results indicate that when a higher voltage is applied a shorter pulse is required to guarantee functionality. A lower V_{op} and a higher t_{op} lead to the best output quality, while the inverse combination achieves the best energy efficiency. We chose the combination $V_{op} = 940\,\mathrm{mV}$ and $t_{op} = 9\,\mu\mathrm{s}$, which retains correct functionality while boasting the highest energy efficiency.

We calculated the average power consumption for each presented adder, where the exact adder requires $491.2686\,\mathrm{pJ}$ per bit. Additionally, $17.455\,\mathrm{pJ}$ are required per input per bit (therefore, $4\times$ for Ax and $5\times$ for the exact versions) to calculate the required complementary values. When we embed the k lowest bits with our Ax adders in an RCA of length n, the resulting energy consumption is:

$$E_{\mathrm{P2AAC}}(k,n) = 274.3175 \cdot k + 578.5436 \cdot (n-k) \quad (7)$$

$$E_{\mathrm{P2AA}}(k,n) = 205.9451 \cdot k + 578.5436 \cdot (n-k) \quad (8)$$

Fig. 5 illustrates the waveforms of the intermediate (NOR) and the output memristors (OR) of P2AAC with the example inputs $a_{i+1,i}$: "01", $b_{i+1,i}$: "11". In the first phase from $0-6\,\mu\mathrm{s}$ the NOR operations calculate the intermediate results. Afterward, the outputs c, s_{i+1}, and s_i are computed in the OR phase from $9-18\,\mu\mathrm{s}$.

C. ERROR METRICS

Since our circuit–level simulations from above match our expected results, we proceed with a behavioral model that provides equivalent functionality from here on out. To quantize and effectively compare the erroneous behavior of Ax full adders, error metrics such as Mean Error Distance (MED), Normalized Mean Error Distance (NMED), and Mean Relative Error Distance (MRED) are commonly used in the SoA [13], [15], [17]. We embed our Ax adders in RCA, vary the approximation degree in steps of two, as the approximations are based on 2–bit units, and evaluate the metrics. The results for 8, 16, and 32–bit are shown in Table IV. We evaluated all 256^2 input combinations for the 8–bit RCA, but chose to only use one million random combinations for 16/32–bit. We did this because a complete evaluation would be very computationally intensive, so a slight stochastic deviation has to be expected. As expected, P2AAC is better than P2AA in every metric, since the carry–out of the 2–bit unit is calculated instead of always being set to '0'. It is important to note that these metrics only represent the general accuracy of an adder but fail to estimate the accuracy for specific applications.

V. CIRCUIT–LEVEL COMPARISON

In this section, we compare our approaches to other SoA logic forms on the most important circuit–level metrics. An overview of exact and Ax adders is shown in Table V. Due to the limited space, we only chose representative adders to represent certain types of approaches. Our adders require almost a magnitude less energy than IMPLY–based approaches. SIXOR is more energy efficient for the full adder, but as it needs TMSL for general logical functions, our approach will require less energy for other circuits. It is important to note that our circuits only require one operational voltage (since it is based on MAGIC and FELIX), while IMPLY and SIXOR need two. Our proposed exact adder is nearly three times as fast as SIXOR and at least four times faster than all other logic forms. The trade–off for these improvements over the SoA is the drastically increased number of memristors and switches. The same trends also apply to our Ax adders P2AAC and P2AA. As our approximations are based on 2–bit units that can be computed in parallel, using an AxD of 4/8, they are 76.3 % – 94.8 % faster and 80.7 % – 87.8 % more energy efficient than the IMPLY–based approximations from [15], [21], [22], [23]. On the other hand, comparable algorithms require 82.5 % – 93.2 % fewer memristors, rendering the trade–off dependent on the application and design goals. The application–level gains of

TABLE IV. Error metrics the proposed approximated adder P2AAC and P2AA for $8/16/32$ bit configurations and with different AxDs.

Full adder	8–bit Adder			16–bit Adder			32–bit Adder		
	MED	NMED	MRED	MED	NMED	MRED	MED	NMED	MRED
	Approximation Degree 2/8			Approximation Degree 4/16			Approximation Degree 8/32		
P2AAC	0.500	9.780e−4	2.754e−3	2.935	2.200e−5	6.200e−5	50.355	< 1.0e−6	< 1.0e−6
P2AA	1.750	3.425e−3	9.434e−3	8.425	6.400e−5	1.770e−4	141.245	< 1.0e−6	< 1.0e−6
	Approximation Degree 4/8			Approximation Degree 8/16			Approximation Degree 16/32		
P2AAC	2.938	5.749e−3	0.016	50.369	3.840e−4	1.068e−3	1.293e4	2.000e−6	4.000e−6
P2AA	8.422	0.016	0.044	141.194	1.077e−3	2.972e−3	3.624e4	4.000e−6	1.200e−5
	Approximation Degree 6/8			Approximation Degree 12/16			Approximation Degree 24/32		
P2AAC	12.441	0.024	0.066	807.990	6.165e−3	0.017	3.311e6	3.850e−4	1.069e−3
P2AA	34.966	0.068	0.163	2.265e3	0.017	0.045	9.281e6	1.080e−3	2.973e−3
	Approximation Degree 8/8			Approximation Degree 16/16			Approximation Degree 32/32		
P2AAC	50.349	0.099	0.244	1.294e4	0.099	0.243	8.480e8	0.099	0.243
P2AA	141.079	0.276	0.508	3.622e4	0.276	0.507	2.375e9	0.277	0.507

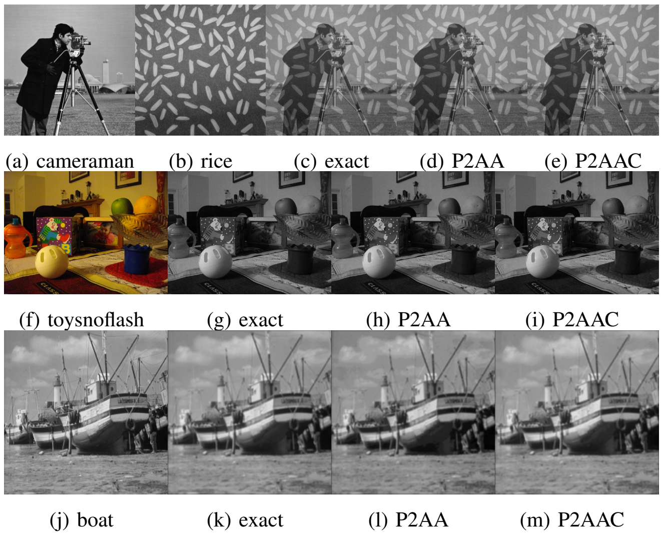


FIGURE 6. Results of different image processing applications with an AxD of $4/8$: image addition (top), grayscale filtering (middle), and Gaussian blurring (bottom).

our Ax adder compared to our exact one will be covered in detail in Section VI and Section VII.

VI. APPLICATION IN IMAGE PROCESSING

A. EXAMPLE APPLICATIONS

Image processing is widely used in computer vision, robotics, medicine, and industrial applications. It is an optimal application for AxC due to its inherent error–resilience. We simulated three common applications and evaluated the PSNR and SSIM, which we show in Fig. 6. In the literature, image addition and grayscale conversion are often used, as they directly showcase the applicability of the Ax full adders. In [15] datasets for both applications were proposed to increase the variety of the experiments. In image addition, two 8–bit images are added together pixel–wise, while in the grayscale filter, the color channels are summed up via two additions [15]. Our results indicate that both P2AA and P2AAC achieve acceptable quality in terms of PSNR with an AxD of up to $4/8$. An example with this AxD is shown in Fig. 6 (a)–(e). We note here that P2AAC is only slightly below the $30\,\mathrm{dB}$ threshold, with an AxD of $6/8$. The experiments of the grayscale filtering lead to roughly the same results, for which an example image is shown in Fig. 6 (f)–(i). To validate the applicability of P2AAC and P2AA in more complex tasks, we utilized our adders to create shift–and–add multipliers and evaluate a Gaussian smoothing example by convoluting a kernel across the image, such as in [15], [35]. We chose this application as it is commonly used to denoise data in computer vision and ML. We used the same $3{\times}3$ kernel as in [15], [36] since it is considered appropriate for 8–bit inputs [35], [36], and applied it to the example image "boat." The results are shown in Fig. 6 (j)–(m), where both Ax adders suffice with an AxD of $4/8$. Our experiments indicate that P2AAC also reaches the $30\,\mathrm{dB}$ PSNR threshold with an AxD of up to $6/8$.

B. APPLICATION–LEVEL GAINS AND COMPARISON

Our presented Ax adders exhibit substantial gains in energy efficiency ($31\,\%$–$40\,\%$) and latency ($50\,\%$) compared to the already efficient exact implementation. In image addition, we can save up to $83.9\,\mu\mathrm{J}$ of energy and up to 393 thousand steps. In the grayscale filter example, up to $291\,\mu\mathrm{J}$ of energy and 1.18 million steps can be saved when our approach is applied. For Gaussian blurring of a 256×256 8–bit image, these gains sum up to $27\,\mathrm{mJ}$ of energy and 88 million steps. Compared to SoA approaches, our adders required $76\,\% - 92\,\%$ fewer steps in image addition while improving the PSNR by $0.1\,\mathrm{dB} - 6.7\,\mathrm{dB}$, when an AxD of $4/8$ is evaluated. For the grayscale filter, our adders improve the PSNR by $2\,\mathrm{dB} - 7.9\,\mathrm{dB}$ with the same gains in latency. A direct energy–efficiency comparison to IMPLY–based approximations may not be entirely fair since they are based on different logic forms. However, we believe it is reasonable to assume that our logic approach is roughly $5 - 8$ times more energy efficient and $5 - 19$ times faster than IMPLY–based circuits. The trade-off for these impressive improvements is an increased footprint, rendering the effectiveness dependent on the intended usage and design goals.

TABLE V. Comparison of key circuit level metrics to SoA exact and approximated full adders.

Full adder	Energy consumption [nJ]		Number of steps		Number of Memristors		Number of Switches	
	n, k	$n=8$ $k=4$	n, k	$n=8$ $k=4$	n, k	$n=8$ $k=4$	n, k	$n=8$ $k=4$
MAGIC [32]A	6.85e$-4n$	5.48e-3	$12n+1$	97	$14n+1$	113	0	0
FELIX [10]A	1.35e$-4n$	1.08e-3	$6n$	48	$4n$	32	0	0
SIXOR [7]B	$0.22n$	1.76	$4n$	32	$9n$	72	0	0
Par. IMPLY [33]B	$4.08n$	32.62	$5n+18$	58	$4n+1$	33	n	8
S–Ser. IMPLY [34]B	$3.84n+0.81$	31.56	$10n+2$	82	$2n+6$	22	12	12
Proposed Exact	$0.58n$	4.63	$3n/2$	12	$53n$	424	$10n$	80
SAFAN [23]	$1.66k+4.83(n-k)$	25.95	$7k+22(n-k)$	116	$2n+3$	19	0	0
PINC [15]	$0.72k+4.08(n-k)$	19.20	$5(n-k)+18$	38	$3k+4(n-k)+1$	29	$n-k$	4
S–SINC [15]	$0.57k+3.84(n-k)$	17.66	$2k+10(n-k)+3$	51	$2n+6$	22	12	12
P2AAC	$0.27k+0.58(n-k)$	3.41	$(3\cdot(n-k)/2)+3$	9	$17k+53(n-k)$	280	$6k+10(n-k)$	64
P2AA	$0.21k+0.58(n-k)$	3.14	$\max(3,(3\cdot(n-k)/2))$	6	$12k+53(n-k)$	260	$4k+10(n-k)$	56

A The presented numbers from MAGIC and FELIX use a theoretical memristor model that renders a direct comparison unfair.
B The values for Par. IMPLY, S–Ser. IMPLY and SAFAN are taken from [15] as they have simulated it with the same memristor model as us.

TABLE VI. Average quality metrics of image processing applications with the proposed P2AAC and P2AA circuits.

Architecture	Image Addition		Grayscale Filter		Gaussian Blurring	
	PSNR [dB]	MSSIM	PSNR [dB]	MSSIM	PSNR [dB]	MSSIM
	Approximation Degree 2/8					
P2AAC	54.236	0.999	48.867	0.997	50.881	0.998
P2AA	46.403	0.995	44.081	0.992	44.317	0.997
	Approximation Degree 4/8					
P2AAC	42.196	0.981	39.715	0.971	45.082	0.993
P2AA	33.375	0.935	31.278	0.911	31.193	0.974
	Approximation Degree 6/8					
P2AAC	29.861	0.828	29.395	0.805	33.517	0.935
P2AA	21.608	0.661	19.681	0.621	19.909	0.811

VII. APPLICATION IN MACHINE LEARNING (ML)

The demand for efficient computation in ML applications necessitates innovative approaches to optimize energy efficiency and inference time. As ML algorithms are inherently tolerant of inaccuracies, P2AAC and P2AA are ideal for drastically increasing performance. With this evaluation, we want to highlight the potential of our proposed methods for ML applications.

A. k–NEAREST NEIGHBORS

To evaluate our circuit on a basic ML application, we chose a classification task based on k–Nearest Neighbors (k–NN). We used the Breast Cancer Wisconsin (Diagnostic) dataset [37] that classifies the data in benign and malignant tumors with a quantization to 8–bit and split $80/20$ into training and test data. For the k–NN we chose $k=3$ with the Manhattan distance metric and a balanced accuracy score. The accuracy and energy consumption of P2AAC and P2AA based RCA with increasing AxD is shown in Fig. 7. Our results indicate that P2AAC is usable for an AxD of 6/16 without accuracy degradation, which saves up to 19.7% of the energy consumption and 33.3% of the steps. P2AA is only applicable with an AxD of up to 2/16, after which the accuracy degrades drastically. This indicates that the error propagation is important for this application.

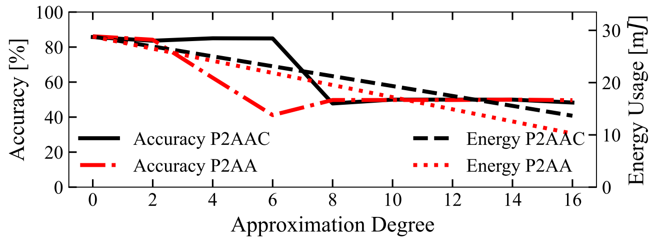


FIGURE 7. Accuracy and energy of the k–NN over an increasing AxD.

TABLE VII. Architecture of the used CNN. Conv: Convolution, FC: Fully Connected, ReLU: Rectified Linear Unit

No.	Type	Kernel	Input	Activation	Output
1	Conv	1×1	$28\times28\times1$	ReLU	$28\times28\times64$
2	Conv	1×1	$28\times28\times64$	ReLU	$28\times28\times32$
3	Conv	1×1	$28\times28\times32$	ReLU	$28\times28\times16$
4	Conv	3×3	$28\times28\times16$	ReLU	$26\times26\times8$
5	Conv	3×3	$26\times26\times8$	ReLU	$24\times24\times4$
6	FC		2304	ReLU	128
7	FC		128	ReLU	64
8	FC		64		10

B. CONVOLUTIONAL NEURAL NETWORK (CNN)

With the rise of deep learning and Neural Network (NN) applications, we wanted to validate the applicability of our Ax adders on a LeNet–5 [38] inspired Convolutional Neural Network (CNN) architecture (Table VII) that was pre–trained on the MNIST [39] training dataset split (60k images). We utilized a partially approximated 16–bit RCA, where either P2AAC or P2AA are used for the lower bits, and embedded it in a shift–and–add multiplier architecture. We quantized the layer inputs and weights to 8–bit signed integer. To evaluate the accuracy and potential gains on inference, we used the 10k test images with varying AxD. Our results indicate, that with an AxD of up to $6/16$, there is no accuracy degradation for both proposed approximations. This is illustrated in Fig. 8, where we save 19.7%–24.1% of energy and 33.3%–37.5% of steps while keeping accuracy. At an AxD of $6/16$, P2AAC saves up to $102.5\,\mathrm{mJ}$ and 505 million steps compared to the exact version. With P2AA, up to $125.6\,\mathrm{mJ}$ and 505 million steps can be saved with no loss of accuracy.

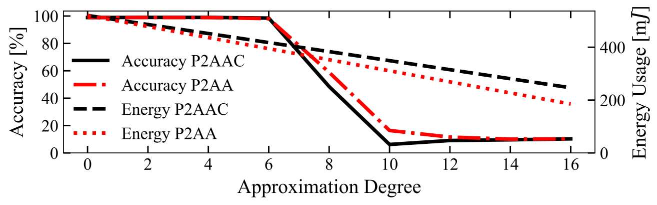


FIGURE 8. Accuracy and energy of the CNN over an increasing AxD.

VIII. CONCLUSION

In this work, we proposed a stateful memristive mixed logic form based on MAGIC and FELIX that is highly flexible, fast, and energy efficient. To showcase the potential of our approach, we implemented a full adder and provided two additional Ax variants that are based on fully parallelizable 2–bit units, further increasing the efficiency. Compared to other memristive logic forms, our proposed exact adders are $62\,\% - 87\,\%$ faster and require up to 85% less energy for an 8–bit addition. We embed our Ax adders in an RCA and evaluate three image processing applications, where we save $50\,\%$ of cycles and $31\,\% - 40\,\%$ of energy compared to our exact adder. We demonstrate the applicability of our approach in ML, where up to $6/16$ bits can be approximated without a reduction in accuracy when tested on a LeNet–5 based CNN that is applied to MNIST. Our approach can save up to $125.6\,\mathrm{mJ}$ and 505 million cycles for each inference.

REFERENCES

[1] M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.

[2] N. TaheriNejad. In-memory computing: Global energy consumption, carbon footprint, technology, and products status quo. pp. 1–6, 2024.

[3] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, 1971.

[4] D. B. Strukov *et al.* The missing memristor found. *Nature*, 453:80–83, 2008.

[5] J. Borghetti *et al.* 'memristive' switches enable 'stateful' logic operations via material implication. *Nature*, 464:873–876, 2010.

[6] E. Lehtonen and M. Laiho. Stateful implication logic with memristors. In *2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 33–36, July 2009.

[7] N. TaheriNejad. Sixor: Single-cycle in-memristor xor. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(5):925–935, 2021.

[8] S. Kvatinsky *et al.* Memristor-based material implication (imply) logic: Design principles and methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2054–2066, 2014.

[9] S. Kvatinsky *et al.* Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[10] S. Gupta *et al.* Felix: Fast and energy-efficient logic in memory. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, 2018.

[11] P. Huang *et al.* Reconfigurable nonvolatile logic operations in resistance switching crossbar array for large-scale circuits. *Advanced Materials*, 28(44):9758–9764, 2016.

[12] W. Liu *et al.* A retrospective and prospective view of approximate computing. *Proceedings of the IEEE*, 108(3):394–399, 2020.

[13] H. Jiang *et al.* Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12):2108–2135, 2020.

[14] H. Jiang *et al.* A review, classification, and comparative evaluation of approximate arithmetic circuits. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13:1 – 34, 2017.

[15] F. Seiler and N. TaheriNejad. Accelerated image processing through imply-based nocarry approximated adders. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2024.

[16] J. J. Yang *et al.* Memristive devices for computing. *Nature nanotechnology*, 8:13–24, 01 2013.

[17] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), mar 2016.

[18] M. S. Ansari *et al.* Low-power approximate multipliers using encoded partial products and approximate compressors. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(3):404–416, 2018.

[19] Z. Wang *et al.* Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[20] C. K. Jha *et al.* Imagin: Library of imply and magic nor-based approximate adders for in-memory computing. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 8(2):68–76, 2022.

[21] S. E. Fatemieh *et al.* Fast and compact serial imply-based approximate full adders applied in image processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):175–188, 2023.

[22] F. Seiler and N. TaheriNejad. An imply-based semi-serial approximate in-memristor adder. In *2023 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pp. 1–7, 2023.

[23] S. Asgari *et al.* Energy-efficient and fast imply-based approximate full adder applying nand gates for image processing. *Computers and Electrical Engineering*, 113:109053, 2024.

[24] N. Kaushik *et al.* High-speed serial and semi-parallel imply-based approximate adders through memristors for in-memory computing. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2024.

[25] D. Radakovits *et al.* Second (v2.0) ltspice implementation of vteam. https://nima.eclectx.org/download/Knowm_Fitted_VTEAM.zip. Last accessed Aug 2024.

[26] S. Kvatinsky *et al.* Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015.

[27] Knowm. Knowm inc. https://knowm.org, 2017.

[28] Knowm Inc. Knowm self directed channel memristors. https://knowm.org/downloads/Knowm_Memristors.pdf. Last accessed Aug 2024.

[29] A. Karimi and A. Rezai. Novel design for a memristor-based full adder using a new imply logic approach. *Journal of Computational Electronics*, 17(3):1303–1314, Sep 2018.

[30] N. Talati *et al.* Logic design within memristive memories using memristor-aided logic (magic). *IEEE Transactions on Nanotechnology*, 15(4):635–650, 2016.

[31] D. Radakovits *et al.* A memristive multiplier using semi-serial imply-based adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(5):1495–1506, 2020.

[32] P. L. Thangkhiew *et al.* Efficient implementation of adder circuits in memristive crossbar array. In *TENCON 2017 - 2017 IEEE Region 10 Conference*, pp. 207–212, 2017.

[33] A. Karimi and A. Rezai. Novel design for a memristor-based full adder using a new imply logic approach. *J. Comput. Electron.*, 17(3):1303–1314, sep 2018.

[34] N. TaheriNejad *et al.* A semi-serial topology for compact and fast imply-based memristive full adders. In *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, 2019.

[35] N. Amirafshar *et al.* Carry disregard approximate multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(12):4840–4853, 2023.

[36] E. Zacharelos *et al.* Approximate recursive multipliers using low power building blocks. *IEEE Transactions on Emerging Topics in Computing*, 10(3):1315–1330, 2022.

[37] W. Wolberg *et al.* Breast cancer wisconsin (diagnostic), 1995.

[38] Y. Lecun *et al.* Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[39] L. Bottou *et al.* Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pp. 77–82 vol.2, 1994.