A Crossbar-Based In-Memory Computing Architecture

Xinxin Wang, *Graduate Student Member, IEEE*, Mohammed A. Zidan, *Member, IEEE*, and Wei D. Lu, *Fellow, IEEE*

***Abstract*— To address the von Neumann bottleneck that leads to both energy and speed degradations, in-memory processing architectures have been proposed as a promising alternative for future computing applications. In this paper, we present an in-memory computing system based on resistive random-access memory (RRAM) crossbar arrays that is reconfigurable and can potentially perform parallel and general computing tasks. The system consists of small look-up tables (LUTs), a memory block, and two search auxiliary blocks, all implemented in the same RRAM crossbar array. External data access and data conversions are eliminated to allow operations fully in-memory. Details of addition, AND logic and multiplication operations are discussed on the basis of search and writeback steps. A compact instruction set consisting of 10 instructions is demonstrated on this architecture through circuit level simulations. Performance evaluations show that the proposed in-memory computing architecture is suitable for handling data-intensive problems. The average power consumption of the crossbar chip is estimated to be 45μW.**

***Index Terms*— In-memory computing, RRAM, crossbar array, look-up table (LUT).**

I. Introduction

THE von Neumann architecture has reigned modern computers for a long period. With Moore's Law leading the way, sequential data processing with buses connecting separated processors and memories has been able to meet the computing needs until recently. However, the "von Neumann bottleneck," i.e., the low speed and high energy demand associated with memory access, has now become the limiting factor of the system performance, a problem magnified in today's big data era [1], [2]. New architectures based on new computing principles and devices are likely required for future computation applications [3], [4].

In-memory computing architectures can efficiently address the von Neumann bottleneck problem and have been extensively studied [5]–[15]. New memory technologies such as resistive random-access memory (RRAM) offer interesting opportunities for computing due to properties of non-volatile storage, high energy efficiency during write/read, high operation speed, and high integration density [16]–[19]. In-memory computing based on RRAM crossbar arrays enables efficient bitwise logic and arithmetic operations, since the different resistance levels can be used to both store data and directly modulate information flow, achieving co-location of memory and logic at the device level. Several previous studies have proposed the implementation of logic circuits using RRAM crossbar arrays with promising performance [5]–[7], [20]. However, significant challenges still remain. For example, the number of operation steps to complete an arithmetic process is typically large. Additionally, input and output data are of different formats, e.g. voltage and resistance values, respectively, requiring external data access and data conversion operations. As a result, data migration will still be present, though the amount of transferred data is reduced.

In this paper, we propose an RRAM-based in-memory and reconfigurable computing architecture that addresses these issues. Both the input and the output data are represented by the devices' resistance values that can be directly accessed and processed, without read-out and write-back. With small look-up tables (LUTs) implemented in the RRAM crossbar, we show the system can be used to process general logic and arithmetic operations. Performance evaluations verify that the proposed in-memory computing architecture offers high efficiency.

II. In-Memory Computing Architecture

The proposed in-memory computing architecture utilizes RRAM crossbar to handle the different computing tasks required for general-purpose computing. Here, we present an entirely in-memory approach, where the data are processed in place without the need for external data access. In this case, the RRAM provides the resources for memory/storage, arithmetic, logic, and data movement within the physical crossbar. This is achieved by virtually splitting the crossbar into four regions: memory/storage region, LUT region, and two auxiliary block regions (search auxiliaries, SAs), as shown in Fig. 1a. The memory storage region stores the inputs and the outputs of the operations. Additionally, it allows dot-product operation to produce the numbers of ONEs in the input vector. Here, we rely on binary RRAM devices to allow the number of ONEs to be counted in a binary fashion. Each device offers two resistance states: the low resistance

Manuscript received March 31, 2020; revised May 25, 2020; accepted May 27, 2020. Date of publication June 18, 2020; date of current version December 1, 2020. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-1617315 and Grant CCF-1900675. This article was recommended by Associate Editor S. Gupta. *(Corresponding author: Wei D. Lu.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: wluee@umich.edu).

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

Digital Object Identifier 10.1109/TCSI.2020.3000468

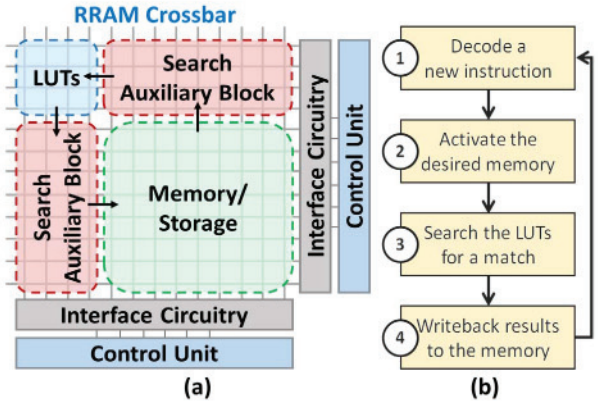

Fig. 1. (a) The proposed in-memory computing architecture with an RRAM crossbar virtually split into four regions. The crossbar is interfaced and controlled using CMOS circuitry. (b) Data flow diagram of the proposed operations.

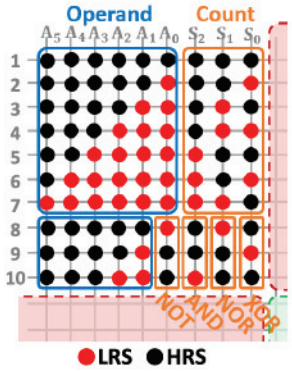

Fig. 2. An example of LUT that implements operations such as count (add), NOT, AND, NOR and XOR by counting the number of ONEs of the input.

state (LRS) representing logic ONE and the high resistance state (HRS) representing logic ZERO. The crossbar array is based on the 1R structure. The top electrodes (TE) of the bipolar RRAM devices are connected to the columns and the bottom electrodes (BE) are connected to the rows. Even though selectors may benefit storage functions, they are not required in the array since all the columns and rows are activated during the computing operations [3].

The other blocks in the system are implemented in the same RRAM crossbar, such that the size and the functions of the blocks are defined by software and can be readily reconfigured. The LUT blocks provide various arithmetic and logic functions for our system, where each LUT block represents a specific arithmetic or logic operation. In general, the presented in-memory computing architecture follows a simple processing cycle, as shown in Fig. 1b. First, each new instruction is decoded by the instruction decoder within the control unit (CU). Accordingly, the desired memory region is activated. With the aid of two search auxiliary blocks (also implemented in the same crossbar), the LUTs are searched for a match for the data under processing. Finally, the result of the LUT search that represents the processed data is written back to the memory portion of the array.

A typical LUT stores a truth table for a given Boolean function. In this case, the table consists of an operand part which stores the binary combinations of n-bit inputs, and the result part which stores their corresponding results. However, such type of LUTs will grow exponentially in size, where the table length equals to 2^n. To address this issue, we note that for standard logic and arithmetic operations, one only care about the number of ONEs per operand, not the specific order of the operand bits. Such approach results in much smaller LUTs, where the length of a table equals to $n+1$ rather than 2^n. Fig. 2 shows examples based on this approach, showing NOT, AND, NOR and XOR as examples of logic function implementation, and counting operation implementation which provides the number of ONEs per operand and is an essential operation of arithmetic multiplication and vector addition. Typically, the LUTs are programmed beforehand and remain static during the computing process. However, the LUT area in the crossbar is assigned through software and can be expanded or shrunk as needed. Designers should know what LUTs they need beforehand and assign specific rows and columns to different parts of the LUT. For example, the count (add) LUT area is assigned by declaring that rows 1∼7 are assigned to it, columns $A_0 \sim A_5$ are assigned to the operands and columns $S_0 \sim S_2$ are assigned to the count results.

The auxiliary blocks can aid the LUT search operation. All devices in this region should be initialized to the HRS and remain unchanged during the operations.

III. Circuit Operation

The proposed computing scheme is based on searching the LUTs for an entry with the same number of ONEs as the input and then write the corresponding result from the desired LUT to the memory block. Below, we detail the circuit operation of different processing steps with the support of HSPICE simulation results, where we adopted the RRAM device model presented in [21]. In this model, the device resistance value depends on the voltage difference across the device. The LRS value is $10^8 \times V(p,n)/\sinh(3 \times V(p,n))$ Ohm and the HRS value is $10^{11} \times V(p,n)/\sinh(3 \times V(p,n))$ Ohm. The set/reset voltages are ± 2V. Assuming 25% variation of the set/reset voltage, in all the simulations in this work we assumed write voltage $V_w = 2.5$V, bias voltage during writeback $V_b = 1$V, and read voltage $V_r = 1$V and pulse width of 50ns. At the read voltage, the device has a LRS of 10MΩ and a HRS of 10GΩ. We consider a 42×41 crossbar where a 32×32 sub-array within it is used as the memory block. In the memory block, different bits of the same data are stored in the same row, with the right most RRAM device representing the lowest bit, and the devices to the left representing higher bits; or in the same column, with the bottom device representing the highest bit, and the upper devices representing lower bits. When processing the bit-wise logic operations, the same bit of two operands are placed in the same row.

A. LUT Search Stage

The first processing step is to find the entry in the LUT that matches the number of ONEs in the input data that is stored in the memory block. Matching the number of ONEs between the memory and the LUT block is done through the SAs,

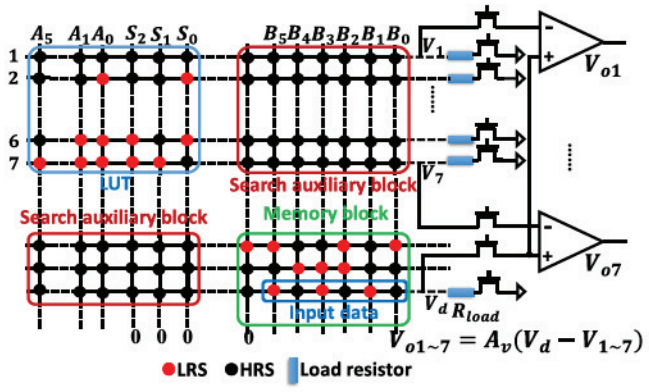

Fig. 3. Demonstration of the LUT search operation. The read voltage is applied to the input columns in the memory block and the operand columns in the LUT. The load resistors convert the output currents into voltage signals, depending on the number of ONEs in the operands. If one row in the LUT contains the same number of ONEs as the operands, the output voltage of the amplifier connected to this row will be closest to zero.

based on the resistance-dependent voltage distribution pattern over the RRAM cells, as shown in Fig. 3. All cells in the SA block are initialized to the HRS and remain unchanged during computing. In this example, a 50ns, 1V read voltage pulse is applied to the input columns $B_0 \rightarrow B_5$ shared by the memory block and a SA block. The same voltage pulses are also applied to the LUT columns $A_0 \rightarrow A_5$, shared by the LUT block and the other SA block, and all the other columns are grounded. Each row in the array is grounded through a series (load) resistor and a NMOS switch. The NMOS switch is turned on with a 2V gate voltage and turned off with a 0V gate voltage. In the example in Fig. 3, row $1 \sim 7$ and the row storing the data (the data row) will be grounded through the on-state switches, all other rows are floated through the off-state switches. The load resistance value is 100KΩ, approximately 100 times lower than the on-state resistance of RRAM device, so that the voltage divider output between the row resistance and the load resistance is directly proportional to the number of ONEs in that row with the switch on. Hence, rows of equal number of ONEs in the memory block and in the LUT block will produce the same voltage. In this case, by comparing the output voltage of the LUT rows with that of the desired memory row, a match can be detected. This can be achieved by implementing sub & amp functions through a group of op-amps, and the LUT row containing the same number of ONEs as the input data will produce the smallest V_o magnitude. Note that there will be no sneak current between column $B_0 \sim B_5$ and $A_0 \sim A_5$ since they have the same electrical potential. Furthermore, as the voltage on the data row is on the order of 10mV, the current in the main path, for example, column B_1-LRS device-data row-load resistor-ground, will be much larger than the current that flows from the grounded columns to the data row. Hence, the sneak current in the search operation can be safely neglected.

After the matched row is identified, the target output can then be written back by copying the value from the LUT to the memory block. In the search operation, the number of op-amps is equal to the number of the LUT rows, and each LUT row is connected to the negative input node of an op-amp, while the data rows are connected to the positive input nodes of all op-amps. An NMOS switch is used between each row and the op-amps. When there are n input data in the search row, the switches (including the ground switches and the op-amp switches) connected to the data row and the first $n+1$ rows in the LUT will be turned on. The remaining switches will be turned off. It should be noted here that a typical memory structure may already require an amplifier per bit-line (BL), so the LUT search amplifiers will not pose an additional overhead to the system. Fig. 4 shows SPICE simulation results of the search process, where all cases from six ZEROs to six ONEs were tested. The crossbar array, the series resistors, the NMOS switches and the amplifiers were included in the simulation. The sneak paths were found to be negligible due to the high ON/OFF ratio and the low load resistance.

B. Results Writeback Stage

Through the search operation, the row matching the input data stored in the memory block is identified. The next step is then to write the LUT results to the memory block. Since the LUT stores elementary functions, they correspond to intermediate results. These results are stored in a region of the memory block that is initialized to HRS (e.g. the intermediate block in Fig. 6). As a result, the writeback of ZEROs will not change any cell status. It will thus improve the computing efficiency by just considering ONEs. As the LUTs are pre-designed, the result values stored in the LUT are associated with their positions. To achieve selective writeback, a 'false' label is attached to each address of ZERO in the LUT. During writeback, the controller would skip this address and perform the next operation when encountering a 'false' label. The writeback of ONEs can be implemented by applying a write voltage pulse to the column and grounding the row corresponding to the target cell (cell T) and applying a bias voltage pulse to the other columns, as shown in Fig. 5(a). The bias voltages can protect the other cells in the array from being programmed in the writeback step when the other rows are left floating. Applying VDD/2 or VDD/3 to the other rows can also achieve this objective. The power consumptions of the VDD/2 scheme is 40.965nW, close to the floating scheme, 42.05nW, while the power consumption of the VDD/3 scheme is 86.094nW, about twice of the floating scheme. Considering these factors, the floating scheme was used to minimize the overhead in the peripheral circuit. Fig. 5(b) plots the voltage signals used in this operation, and Fig. 5(c) shows the writeback results, with cell T set to LRS. The bias voltage can protect other devices in the array from being updated, so the sneak paths during writeback can be mitigated as long as the array size is not too large where the line resistance becomes significant.

Note that in the proposed in-memory computing approach, the input and output data are both resistance values and there is no data movement between the "memory" and the "computing" units. The conversion from the input data to an output corresponding to the number of ONEs (based on different output voltage levels) is effectively a dot-product operation on the RRAM devices, which is common in RRAM/memristor-based neural network implementations [3].

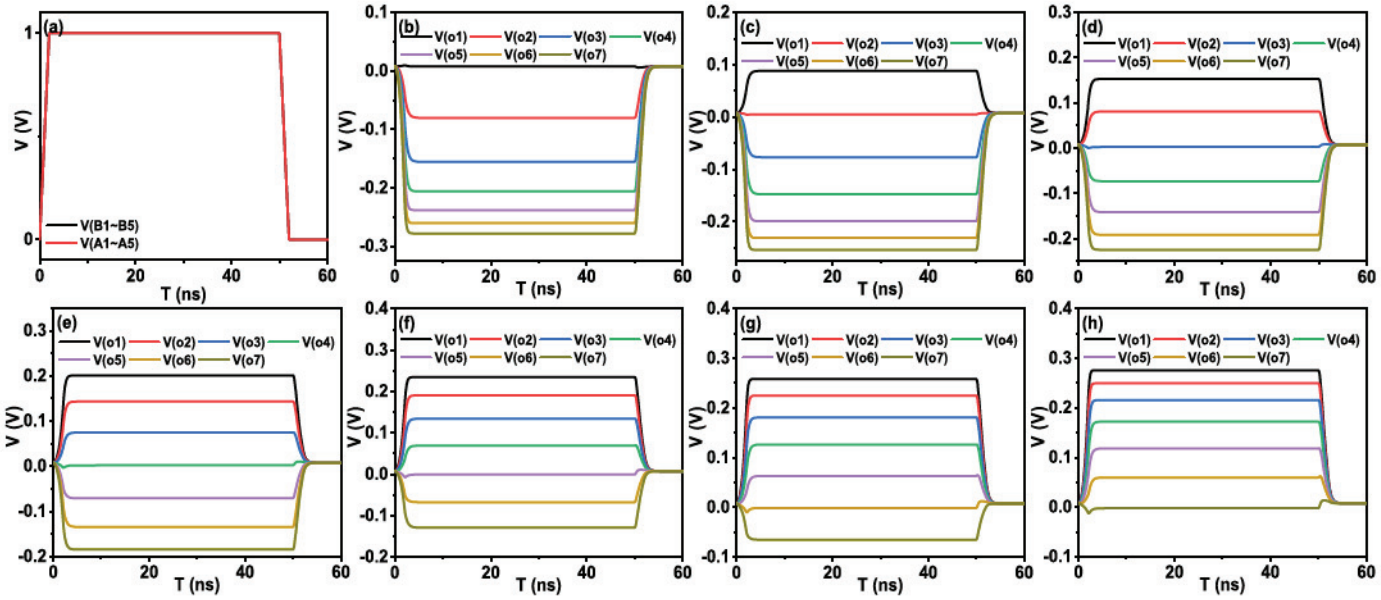

Fig. 4. (a) Read pulses used in the simulation, and (b)∼(h) simulation results of the search stage, for cases from six ZEROs to six ONEs, respectively. The curves $V_{O1} \sim V_{O7}$ represent the output voltage signals of the amplifiers for LUT row 1 to row 7.

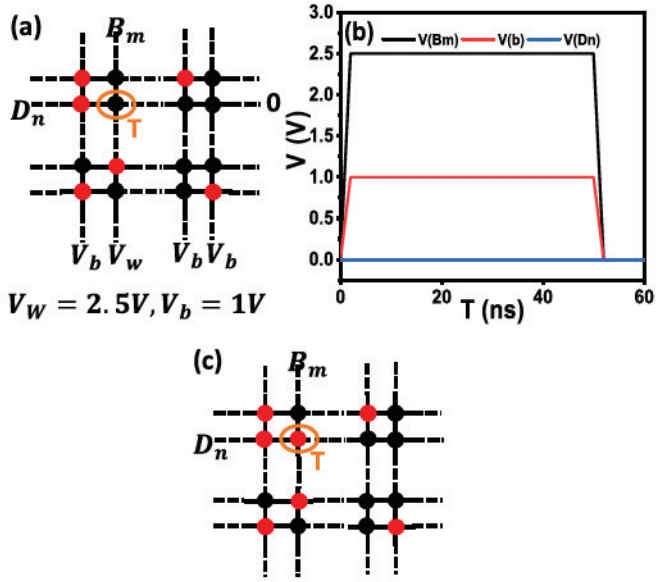

Fig. 5. (a) Example of writeback ONE to cell T in the intermediate block. (b) Input voltage signals applied during the writeback process. (c) Result of the writeback process.

IV. Arithmetic and Logic Functions

A. In-Memory Vector Operations

One of the major strengths of in-memory computing is the native ability to process large batches of data in a fully parallel manner, commonly in vector form. Here, we demonstrate how the presented architecture can perform tree reduction, which is a core operation for multiplication and vector addition [20]. In this case, the result of the tree reduction is a function of the number of ONEs for a given vector. Hence, we adopted the LUT to produce the number of ONEs, as discussed earlier. The desired results obtained from the LUT is subsequently stored back in the intermediate block and used to produce the final output in the memory.

B. Addition

An example of vector addition operation is shown in Fig. 6. In this example, we compute the sum of a four-operand vector, where each element is made of four bits. As the vector length is four, there are at most six input operands in each bit, and therefore the LUT in the array contains seven rows. At the very beginning, a small block (the intermediate block) is allocated and initialized to HRS to facilitate the writeback of intermediate results. The addition is performed by counting the number of ONEs through each row that represents a single bit precision. For instance, the first row (A_0, B_0, C_0, D_0) is counted through the LUT and the result is stored in cells S_{00}, S_{01}, and S_{02}. New counts are performed after this update for the other rows (bits) including previous results, until full addition process is completed. If the result should be stored in the search row, it is directly written into the cell in the final result column, otherwise the result is always written into the left cell adjacent to the previous intermediate data. Note that the results always contain more bits than the operands, so the steps mentioned above should be repeated more than 4 times. For example, the steps are repeated six times for the addition of four 4-bit numbers.

C. Logic Gates

By utilizing LUTs, different logic operations can be implemented in the presented in-memory computing approach. Here, we consider standard logic gates with symmetrical inputs. In this case, the LUTs can be simply searched with the number of ONEs per input, as discussed before. As an example, Fig. 7 shows a demonstration for an AND gate operation supported by SPICE simulation results. In Fig. 7(a), columns A_1 and A_2 correspond to the operand part and column S_2 represents the result part of the LUT block. Fig. 7(b)∼(d) show results from the read operation of three different input combinations: 00, 01 and 11. Other logic

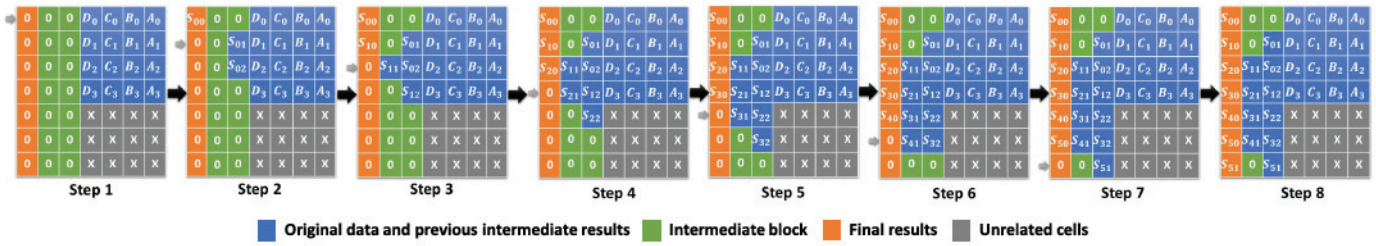

Fig. 6. Process flow of the addition of four 4-bit numbers (A, B, C, D). The search operation finds an entry in the LUT that matches the number of ONEs in the input. The write operation subsequently writes the desired value into the target cell. Blue cells represent operands of each step, green cells represent the cells reserved for writing results of each bit and orange cells represent the final results.

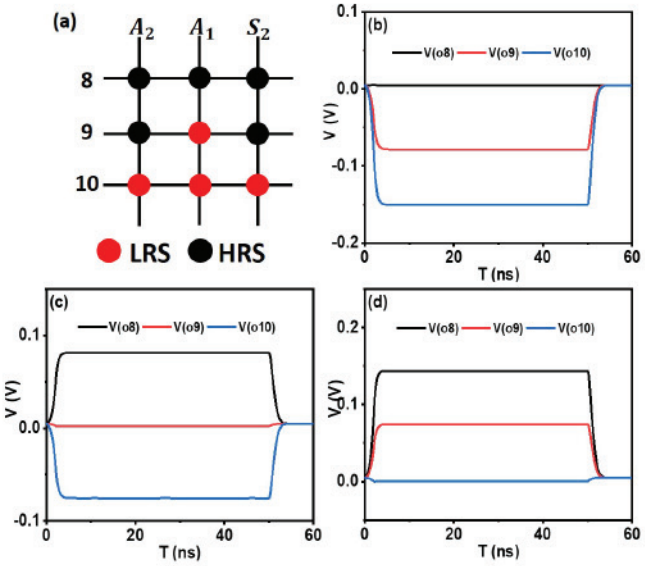

Fig. 7. (a) The LUT for AND gate. (b)∼(d) Amplifier outputs corresponding to different input combinations, 00, 01, and 11, respectively.

operations can be implemented with the assistance of different LUTs.

D. Multiplication

Typically, multiplication is done by adding the different partial products together in a multi-operand addition operation [20]. In these approaches, the addition is performed sequentially elementwise. In contrast, our proposed approach can achieve an in-memory multiplication through vector addition combined with bitwise AND, which can reduce the number of operation steps required. An example of the multiplication of two 4-bit numbers (1101 × 1011) is demonstrated in Fig. 8. At the very beginning, the two input numbers are stored in eight different cells in the same row. A 9 × 6 block within the memory block is allocated to store the intermediate results and the final result. The first part of the multiplication is four 4-bit AND operations. Fig. 8 exhibits an example of the first operation, $B_4B_5B_6B_7$ AND $B_0B_0B_0B_0$ (1101 AND 1111) which is executed between 0∼700ns and the result 1101 is writeback to the first four cells in column B_0. The next three 4-bit AND operations are performed in the following 1800ns. The time consumed varies with different number of writeback steps. The second part is the addition. The search and writeback operations are performed bit by bit. An example of the write back of intermediate values and the updated results can be found in Fig. 8, from 2500ns to 2900ns. For the arithmetic operations, the execution is terminated at the bit which contains only one input number, at 4400ns.

E. Parallel Computing

Parallel arithmetic and logic processing on this proposed architecture can be achieved in a single instruction, multiple data (SIMD) manner. Fig. 9(a) shows the circuit implementation for parallel searching. N groups of input data should be stored on the same columns and *n* different rows. There are *n* groups of op-amps and each group contains the same number of op-amps as the LUTs rows. For every entry in the LUT, the negative input nodes of *n* op-amps are connected to the corresponding row in the LUT, and the positive input nodes are connected to the *n* different data rows. These op-amps produce the comparison results of the input data to the LUT data in parallel. For each parallel writeback operation, the target cells of different groups of data should be on the same column. As Fig. 9(b) shows, the parallel writing of cell T_0 and T_n is similar to the writing of a single device as discussed in section III, the only difference is that multiple rows, rather than a single one, are grounded. Using the parallel approach can greatly speed up computing. For example, 4-bit logic gates will consume one initiate, four search and on average two writeback steps in bit-by-bit computing, but will only consume three steps: one initiate, one search and one writeback in parallel computing.

V. Performance Estimate

In this section we discuss the performance evaluation results of the proposed system. The average power consumption, energy consumption and number of operation steps for vector addition were analyzed through simulation, including effects from the crossbar array, the load resistors, the NMOS that works as switches and the op-amps that perform comparisons. The average power of the system based on a 42 × 41 crossbar array as a 4-bit processor was also estimated. Since in-memory processing approaches in general target vector operations, we use the vector addition function as our reference operation. The performance evaluation results of vector addition are shown in Fig. 10. As the memory block has 32 rows, the vector length ranges from 2 to 28 in this analysis so that the number of ONEs in each row will not exceed 32. The average number of operation steps is obtained by directly counting the average number of search steps and writeback

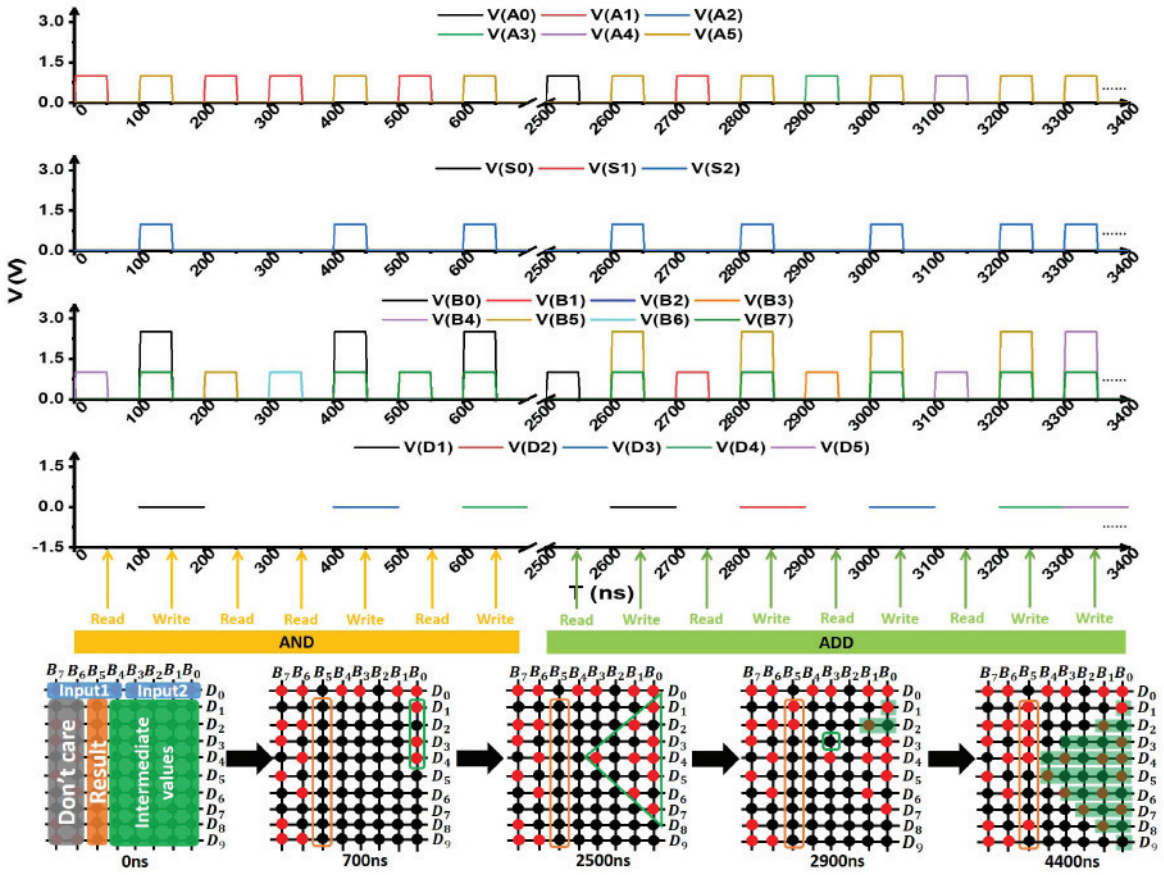

Fig. 8. An example for multiplication (1101 × 1011), showing the voltage signals applied to the rows and columns and the device resistance states at different operation stage. At the beginning of the execution, an intermediate block and the region to store the final results are allocated and initialized to HRS. During the execution, the AND operation is performed first and the results are stored in the green triangle region of the intermediate block. In this example, $B_4B_5B_6B_7$ AND $B_0B_0B_0B_0$ takes 700ns and the total AND operation takes 2500ns. In the following ADD operation, the search and write operation are performed row by row, starting from row D_1. For the first two rows, there are two search and two writeback operations, consuming 400ns. The total execution is terminated at the row which only contains one input data, at 4400ns.

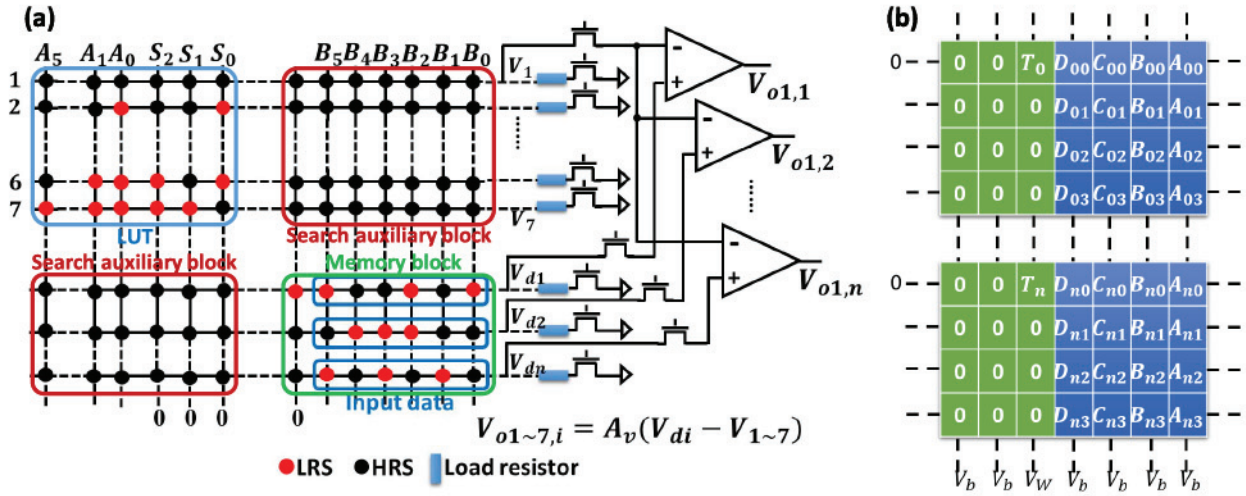

Fig. 9. Circuit implementation for (a) parallel searching and (b) parallel writing.

steps, shown in Fig. 10(a). The change of total number of cycles with the vector length is nearly a step function. With longer vector there could be more ONEs under the worst case, leading to wider bit width for the sum of each bit and therefore more read and write steps for the desired function. The growth rate is however sublinear since the number of bits in the addition result increases sublinearly with vector length. In contrast, in previous approaches this parameter usually grows linearly with the number of operands, resulting in low operation speed [8].

The power and energy consumption for 4-bit vector addition were also analyzed, as shown in Fig. 10(b) and (c), respectively. The power consumption is calculated as the arithmetic average of each step, including initiation, search and

TABLE I
PERFORMANCE ESTIMATION OF THE PROPOSED ARCHITECTURE AND COMPARISON WITH OTHER LITERATURE RESULTS

Instruction	Average cycles			Average power (μW)	Average Energy (pJ)	Area (number of RRAM devices)	Throughput (MOPS)
	Initiate	Search	Writeback				
add	1	5	4	37	18	18	6
	1 [7]	0 [7]	40 [7]	7 [7]	14 [7]	33 [7]	
						40 [28]	
dot/and	1	1	1	55	8	12	26.66
	1 [7]	0 [7]	1 [7]	24 [7]	2.4 [7]	12 [7]	
	1 [32]	0 [32]	1 [32]	23 [32]	2.3 [32]	12 [32]	
mul	1	25	18	41	90	62	0.68
	1 [7]	0 [7]	116 [7]	8 [7]	45 [7]	88 [7]	
						4109 [29]	
sub	1	5	5	45	25	26	5.45
	1 [7]	0 [7]	44 [7]	6 [7]	14 [7]	37 [7]	
shift	1	1	1	54	8	8	26.66
mask	1	1	1	54	8	8	26.66
mov	1	1	1	54	8	8	26.66
not	1	1	1	54	8	8	26.66
	1 [7]	0 [7]	1 [7]	23 [7]	2.3 [7]	8 [7]	
	1 [32]	0 [32]	1 [32]	23 [32]	2.3 [32]	8 [32]	
xor	1	1	1	63	9	12	26.66
	1 [7]	0 [7]	5 [7]	38 [7]	11 [7]	28 [7]	
nor	1	1	1	63	9	12	26.66
	1 [7]	0 [7]	3 [7]	34 [7]	7 [7]	20 [7]	
	1 [32]	0 [32]	1 [32]	24 [32]	2.4 [32]	12 [32]	

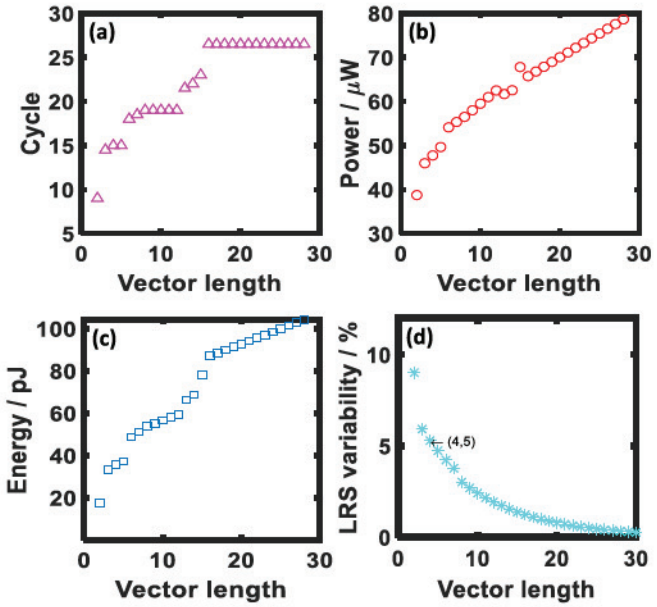

Fig. 10. Benchmarking results of 4-bit vector addition. (a) Average number of cycles demanded for the addition. (b) Power consumption and (c) energy consumption of the array and amplifiers. (d) Tolerance of LRS variation.

writeback operations. The results show that the array consumes more power and energy with longer vector length, while the growth rates are also sublinear.

In a more general approach, we analyzed the in-memory architecture performance for a simple instruction set architecture (ISA), including add, dot (and), mul, sub, shift, mask, mov, not, xor and nor. In this analysis, all instructions are based on 4-bit numbers, and the logic, shift, mask and move operations are performed in a parallel fashion. The average power consumption of initiation, search and writeback operations were simulated respectively. The number of operation steps for different instructions were counted. Then the average power and energy of each instruction were calculated, assuming 50ns pulse width during each step. The areas required for executions are represented by the number of RRAM devices included in the computing. Finally, the throughput of this small crossbar chip is stated in units of mega operations per second (MOPS), assuming parallel operation of the inputs. The results are shown in Table 1. Comparison with recently reported in-memory computing architectures is also provided in the table, in which the average power of [7] is obtained using the reported architecture and the device model used in this study. The approaches discussed in [7] and [32] can offer very good power and energy performance, but these functions are implemented by programming output RRAM devices based on the voltage divider effect and are sensitive to sneak paths which can severely limit the system's capabilities to deliver the required programming voltage and current during write [37]. In the proposed approach, the voltage divider effect is only used during the search operation (i.e. read operation). As discussed in III.A, the sneak current problem can be effectively mitigated with proper biasing schemes during read. In general, the proposed approach is more efficient for executing complex logic and arithmetic operations and can effectively circumvent the sneak path problem which constraints previous implementations of RRAM-based ALU in large arrays. Furthermore, by assuming that chances of executing the 10 instructions in Table 1 are equal, we can calculate the average energy cost for a single instruction (19.1pJ) by dividing the sum of the

average energy cost with the number of instructions. We note the energy cost in CMOS based technologies is dominated by memory access. For example, Ref. [38] shows that the energy cost of a DRAM access in 40nm CMOS-based computing architecture is 10pJ/bit, suggesting CMOS microprocessors at this technology node will consume more than 40pJ for 4-bit operations even without considering energy consumption of the arithmetic logical unit (ALU). Similarly, we can estimate the average power of the proposed architecture to be 45μW, by dividing the sum of the average energy by the sum of the average number of cycles to perform the instructions.

In terms of the area, the op-amps and control circuit can both add overhead to the system. The op-amp consists of 11 MOSFETs. The area of one op-amp is approximately $0.08\mu m^2$ in 65nm technology. Assuming the area of an RRAM device is $4F^2$, the area of the crossbar array is $29.1\mu m^2$. The op-amp area overhead is 11.5% if there is one op-amp connected to each row. The control circuit is a key part of the in-memory computing system. The controller may be complex and can introduce a large area overhead when many concurrent data need to be read/write simultaneously on the memory array. However, the proposed architecture only requires one read or write operation per cycle. Following [10], the controller can be limited to a simple finite state machine (FSM) and a few registers in this case.

To make the results more realistic, we need to consider the impact of non-ideal effects, such as conductance variations and line resistance. We first evaluated the demand of conductance uniformity in the LUT search stage. HRS resistance variation has negligible impact on the read current. However, large LRS resistance variation can lead to incorrect results of counting ONEs. To ensure the validity of the search results, LRS variation should be reduced as the vector length grows, as Fig. 10(d) shows. Typically, to perform addition of four 4-bit numbers, the LRS variation should be no larger than 5%. The line resistance effect was also analyzed and was found to have a minimal effect. As stated in [7], the BL and WL resistance of a 16×8 array is smaller than 50Ω. Thus, for the proposed 42×41 array the line resistance is expected to be smaller than 260Ω, much lower than the load resistance value of $100K\Omega$.

Notably, the proposed architecture also has limitations. The LUTs do not support all functions. When it comes to instructions in which the sequence of inputs can affect the results, such as implication, or complex instructions such as divide and floating-point operations, computing should be performed as the combination of a series of simple instructions.

VI. Conclusion

In this paper, we proposed an in-memory parallel processing architecture and presented the detailed array implementation and process flow. As examples of arithmetic and logic operations, vector addition, vector multiplication and AND logic are performed using the system. The architecture is reconfigurable since different LUTs can be integrated in the RRAM array, while the different blocks can be freely changed since their functions are solely defined in software. Data migration outside the crossbar is eliminated, as there is no need to know any resistance value during the complete operation process. The operation of a simple ISA on the proposed architecture has been verified through circuit-level simulations with high speed and efficiency.

References

[1] P. Kogge *et al.*, "ExaScale computing study: Technology challenges in achieving ExaScale systems," Defense Adv. Res. Projects Agency Inf., Arlington County, VI, USA, Tech. Rep., 2008, vol. 15.

[2] R. Nair, "Evolution of memory architecture," *Proc. IEEE*, vol. 103, no. 8, pp. 1331–1345, Aug. 2015.

[3] M. A. Zidan, A. Chen, G. Indiveri, and W. D. Lu, "Memristive computing devices and applications," *J. Electroceram.*, vol. 39, nos. 1–4, pp. 4–20, Dec. 2017.

[4] M. M. Shulaker *et al.*, "Three-dimensional integration of nanotechnologies for computing and data storage on a single chip," *Nature*, vol. 547, no. 7661, pp. 74–78, Jul. 2017.

[5] B. Chen, F. Cai, J. Zhou, W. Ma, P. Sheridan, and W. D. Lu, "Efficient in-memory computing architecture based on crossbar arrays," in *IEDM Tech. Dig.*, Dec. 2015, p. 17.5.

[6] H. Li *et al.*, "A learnable parallel processing architecture towards unity of memory and computing," *Sci. Rep.*, vol. 5, no. 1, Oct. 2015, Art. no. 13330.

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

[8] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar, "Resistive GP-SIMD processing-in-memory," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, p. 57, 2016.

[9] G. Papandroulidakis, I. Vourkas, A. Abusleme, G. C. Sirakoulis, and A. Rubio, "Crossbar-based memristive logic-in-memory architecture," *IEEE Trans. Nanotechnol.*, vol. 16, no. 3, pp. 491–501, May 2017.

[10] P.-E. Gaillardon *et al.*, "The Programmable Logic-in-Memory (PLiM) computer," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 427–432.

[11] Y. Yang, J. Mathew, S. Pontarelli, M. Ottavi, and D. K. Pradhan, "Complementary resistive switch-based arithmetic logic implementations using material implication," *IEEE Trans. Nanotechnol.*, vol. 15, no. 1, pp. 94–108, Jan. 2016.

[12] H. A. D. Nguyen, L. Xie, M. Taouil, R. Nane, S. Hamdioui, and K. Bertels, "Computation-in-memory based parallel adder," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2015, pp. 57–62.

[13] H. Jarollahi *et al.*, "A nonvolatile associative memory-based context-driven search engine using 90 nm CMOS/MTJ-hybrid logic-in-memory architecture," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 4, no. 4, pp. 460–474, Dec. 2014.

[14] L. Xie, H. A. D. Nguyen, M. Taouil, and K. B. S. Hamdioui, "Fast Boolean logic mapped on memristor crossbar," in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2015, pp. 335–342.

[15] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2015, pp. 336–348.

[16] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electron.*, vol. 1, no. 1, pp. 22–29, Jan. 2018.

[17] P. Sheridan and W. Lu, "Memristors and memristive devices for neuromorphic computing," in *Memristor Networks*. Cham, Switzerland: Springer, 2014, pp. 129–149.

[18] H.-S. P. Wong et al, "Metal-oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

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

[20] M. A. Zidan, Y. Jeong, J. H. Shin, C. Du, Z. Zhang, and W. D. Lu, "Field-programmable crossbar array (FPCA) for reconfigurable computing," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 4, pp. 698–710, Oct. 2018.

[21] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology*, vol. 20, no. 42, Oct. 2009, Art. no. 425204.

[22] S. H. Jo and W. Lu, "CMOS compatible nanoscale nonvolatile resistance switching memory," *Nano Lett.*, vol. 8, no. 2, pp. 392–397, Feb. 2008.

[23] H. Li *et al.*, "Statistical assessment methodology for the design and optimization of cross-point RRAM arrays," in *Proc. IEEE 6th Int. Memory Workshop (IMW)*, May 2014, pp. 1–4.
[24] R. Liu *et al.*, "Impact of pulse rise time on programming of cross-point RRAM arrays," in *Proc. Tech. Program Int. Symp. VLSI Technol., Syst. Appl. (VLSI-TSA)*, Apr. 2014, pp. 1–2.
[25] L. Magnelli, F. A. Amoroso, F. Crupi, G. Cappuccino, and G. Iannaccone, "Design of a 75-nW, 0.5-V subthreshold complementary metal-oxide-semiconductor operational amplifier," *Int. J. Circuit Theory Appl.*, vol. 42, no. 9, pp. 967–977, Sep. 2014.
[26] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 770–775.
[27] D. Chakraborty, S. Raj, and S. K. Jha, "A compact 8-bit adder design using in-memory memristive computing: Towards solving the Feynman grand prize challenge," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architectures (NANOARCH)*, Jul. 2017, pp. 67–72.
[28] D. Chakraborty and S. K. Jha, "Design of compact memristive in-memory computing systems using model counting," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[29] A. Ul Hassen, D. Chakraborty, and S. K. Jha, "Free binary decision diagram-based synthesis of compact crossbars for in-memory computing," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 5, pp. 622–626, May 2018.
[30] A. Ul Hassen, S. A. Khokhar, H. A. Butt, and S. K. Jha, "Free BDD based CAD of compact memristor crossbars for in-memory computing," in *Proc. 14th IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2018, pp. 1–7.
[31] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Interconnect networks for memristor crossbar," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architectures (NANOARCH)*, Jul. 2015, pp. 124–129.
[32] S. Kvatinsky *et al.*, "MAGIC—Memristor-aided logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.
[33] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2015, pp. 1718–1725.
[34] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, Jul. 2016.
[35] A. Flocke and T. G. Noll, "Fundamental analysis of resistive nano-crossbars for the use in hybrid nano/CMOS-memory," in *Proc. 33rd Eur. Solid-State Circuits Conf. (ESSCIRC)*, Sep. 2007, pp. 328–331.
[36] A. Flocke, T. G. Noll, C. Kugeler, C. Nauenheim, and R. Waser, "A fundamental analysis of nano-crossbars with non-linear switching materials and its impact on TiO_2 as a resistive layer," in *Proc. 8th IEEE Conf. Nanotechnol.*, Aug. 2008, pp. 319–322.
[37] S. Kim, H.-D. Kim, and S.-J. Choi, "Numerical study of read scheme in one-selector one-resistor crossbar array," *Solid-State Electron.*, vol. 114, pp. 80–86, Dec. 2015.
[38] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2014, pp. 10–14.

Xinxin Wang (Graduate Student Member, IEEE) received the B.S. degree in microelectronics science and engineering from Peking University in 2018. She is currently pursuing the Ph.D. degree with Electrical Engineering and Computer Science Department, University of Michigan. Her research interest includes hardware neural network accelerator design based on resistive random access memory (RRAM) crossbar arrays.

Mohammed A. Zidan (Member, IEEE) received the B.Sc. degree (Hons.) in electronics and communications engineering from the Institute of Aviation Engineering and Technology (IAET) in 2006, the M.Sc. degree (Hons.) in electronics and communications engineering from Cairo University in 2010, and the Ph.D. degree in electrical engineering from the King Abdullah University of Science and Technology (KAUST), Saudi Arabia, in 2015, with a GPA of 4.0. He is currently a Post-Doctoral Fellow with the University of Michigan, Ann Arbor. His research interests include neuromorphic/in-memory computing, RRAM circuits and systems, and computer arithmetic. He was a recipient of the IEEE Circuits and Systems (CAS) Society Pre-Doctoral Scholarship Award.

Wei D. Lu (Fellow, IEEE) received the B.S. degree in physics from Tsinghua University in 1996 and the Ph.D. degree in physics from Rice University in 2003. He was a Post-Doctoral Research Fellow with Harvard University from 2003 to 2005. He joined the faculty of the University of Michigan in 2005. He is a Professor with Electrical Engineering and Computer Science Department, University of Michigan. His research interests include resistive-random access memory (RRAM)/memristor devices, neuromorphic systems, aggressively scaled transistor devices, and low-dimensional systems.