Carry-free Addition in Resistive RAM Array: n -bit Addition in 22 Memory Cycles

John Reuben

Chair of Computer Architecture

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
91058 Erlangen, Germany
johnreuben.prabakar@fau.de

Dietmar Fey

Chair of Computer Architecture

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
91058 Erlangen, Germany
dietmar.fey@fau.de

Abstract—The movement of data between processing and memory units, often referred to as the ‘von Neumann bottleneck’ is the main reason for the degraded performance of contemporary computing systems. In an effort to overcome this bottleneck, methods to ‘compute’ at the location of data are being pursued in many emerging memories, including Resistive RAM (ReRAM). Although many prior works have pursued addition in memory, the latency of n -bit addition has not been judiciously optimized, resulting in $O(n)$ or at best $O(\log(n))$. Computing with three states can enable carry-free addition and result in a latency which is independent of operand width ($O(1)$). In this work, we propose a method to perform carry-free addition completely in memory (a storage array, a processing array and their peripheral circuitry). The proposed technique incurs a latency of 22 memory cycles, which outperforms other in-memory binary adders for $n \geq 32$. This speed is achieved at the cost of increased peripheral hardware.

Index Terms—Non-Volatile Memory (NVM), memristor, Resistive RAM, 1 Transistor-1 Resistor (1T1R), von Neumann bottleneck, in-memory computing, stored-transfer representation, ternary computing, carry-free addition

I. INTRODUCTION

The movement of data between processing and memory units is the major cause for the degraded performance of contemporary computing systems, often referred to as the ‘von Neumann bottleneck’ or ‘memory wall’. ‘Computation energy’ is dominated by ‘data movement energy’ since the energy for memory access grows exponentially along the memory hierarchy (from cache to off-chip DRAM). There has been an ongoing effort (for 15-20 years) to combat the memory wall by bringing the processor and memory unit closer to each other. For example, 3D stacking of DRAM dies over logic die (enabled by Through-Silicon-Via technology) was used to reduce the energy and latency of data movement between processor and memory, in what was called near-memory computing [1]. Going a step further, efforts are being made to move computing not just near memory, but to the memory itself *i.e.*, the memory array.

Resistive RAMs (ReRAMs) are two terminal devices (usually a Metal-Insulator-Metal structure) capable of storing data as resistance. When subject to voltage stress, its resistance can be switched reversibly between a Low Resistance State (LRS) and a High Resistance State (HRS). The change of resistance is due to the formation or rupture of a conductive filament in

the insulator, depending on the direction of the current flow through the structure. The word ‘memristor’ is also used by researchers to denote such a device, because it is essentially a resistor with memory. The word memristor and Resistive RAM are used interchangeably in this work. However, it must be noted that the word memristor can refer to a broader class of devices which have the capability to change their resistance in response to voltage/current stress (*e.g.* Phase Change Memory (PCM), Spin Transfer Torque-Magnetic RAM (STT-MRAM)).

Although ReRAM (memristor) was initially experimented as a non-volatile memory technology, it was later discovered that certain Boolean logic operations (IMPLY, NOR, NAND [2]) can be implemented in the memory array. Boolean gates were implemented by modifying the structure of the memory array or modifying the peripheral circuitry or a combination of these. Arithmetic operations like addition/subtraction were implemented as a sequence of Boolean NAND/NOR/IMPLY operations. It was found that if arithmetic operations can be implemented in memory, the ‘memory wall’ could be overcome since the costly data transfer (both energy and latency-wise) between processor and memory units is eliminated. This paradigm shift in the way the data is processed heralded a new era in computing – ‘in-memory computing’ or ‘processing-in-memory’ or ‘compute-in-memory’. A survey of research motivated by this paradigm can be found in [3], [4]. It must be noted that the term ‘in-memory computing’ may also refer to cognitive tasks like machine learning and pattern recognition performed in memory. In this work, our focus is ‘in-memory arithmetic’.

Although there had been a plethora of works on in-memory arithmetic, it is evident that latency of such in-memory adders has not been carefully studied and optimized. As a result, they require hundreds of cycles to perform 32-bit addition in memory [5]. This exorbitant latency ($O(n)$ for adding two n -bit numbers) can be attributed to rippling of carry and weak logic primitives used (IMPLY/NOR). To overcome this latency hurdle, two paths were pursued—stronger logic primitives and parallel-prefix configurations. Majority logic primitive proved to be stronger than NAND/NOR/IMPLY primitives making in-memory addition fast [6]–[8]. To avoid rippling of carry, parallel-prefix adders were investigated. Parallel-prefix adders could reduce the latency of in-memory addition to

$8 \cdot O(\log(n))$ and $4 \cdot O(\log(n))$ using OR/AND primitives [9] and MAJORITY/NOT [5], respectively.

The conventional approach to addition in both CMOS technology and in emerging non-volatile memories is binary – operands are represented in binary format and processed in binary to get the sum in binary. In this work, we take a different approach to tackle the exorbitant latency of in-memory addition. It has long been known that computing with three states enables ‘carry-free addition’ in which two operands with wordlength of n can be added in constant time, *i.e.*, independent of n in $O(1)$, whereas binary adders can perform that only in $O(\log(n))$ steps, if reasonable hardware resources are used. In [10], usefulness of multi-level cell memristive devices for ternary computing was shown for the first time. In [11], [12], carry-free adders were implemented in a hybrid manner – ReRAMs were used for storing ternary values but were processed in conventional CMOS after analog-to-digital conversion (ADC). The resulting digit after CMOS processing had to go through digital-to-analog conversion (DAC) before they are stored in ReRAMs. This ADC and DAC costs energy and latency, making carry-free addition less efficient. In this work, we propose a methodology to implement carry-free addition completely in non-volatile memory (a ReRAM array where ternary data are stored, a processing array and peripheral circuitry of the arrays).

The rest of the paper is organized as follows. In Section II, we introduce Stored-Transfer Number Representation (STNR) which lays the foundation for carry-free addition. The methodology to implement carry-free addition in memory is presented in Section III. Section IV elaborates the detailed circuit-level implementation of carry-free addition and the verification of the circuit’s functionality by simulation. We compare the proposed addition with state-of-the-art in section V followed by conclusion (Section VI).

II. STORED-TRANSFER REPRESENTATION AND CARRY-FREE ADDITION

The idea of redundant number representations goes back to the 1950s and 1960s [13], [14] when electronic circuit integration technology was not so strongly developed as today and one was forced to save transistors as much as possible for the realization of fast arithmetic circuits, *e.g.* adders. By using redundancy in one digit either by introducing a -1 in addition to the binary values 1 and 0, called balanced ternary representations (digit $d_i \in \{-1, 0, 1\}$), or by introducing a 2 in addition to 0 and 1 in Stored-Transfer Number Representation (digit $d_i \in \{0, 1, 2\}$), one could avoid time-consuming carry chains while adding two operands. The idea of using the ternary digits 0, 1, and 2 for carry-free addition was first published in 1959 by [13]. Later, a full adder solution for Signed Digit (SD), a type of redundant number representation, was published by Jaberipur et.al. in 2006 [15]. However, without loss of generality, we focus here on a method proposed by us to perform carry-free addition using STNR. Consider two ternary numbers $X, Y \in \{0, 1, 2\}$, as depicted in Fig. 1. They can be added in a carry-free manner in three stages.

In each stage, we calculate a sum digit Z_i and a transfer digit T_{i+1} according to the truth tables depicted for the corresponding stage. The transfer digit is denoted T_{i+1} because for a particular digit position i , the transfer digit is similar to the ‘carry’ propagated to the next digit. But unlike the carry propagated in conventional adders, the transfer digit T_{i+1} can be computed without waiting for the lower digits, *i.e.*, for 8-digit addition, T_5 depends solely on X_4, Y_4 and NOT on $X_{4,3,2,1,0}, Y_{4,3,2,1,0}$. This attribute of T_{i+1} implies that, for n -digit addition, all n T_{i+1} values can be computed in parallel. Z^1 and T^1 are calculated in parallel from X, Y in stage 1 following the rule,

$$X_i + Y_i = 2 \times T_{i+1}^1 + Z_i^1 \quad (1)$$

In stage 2, Z^2 and T^2 are calculated in parallel from Z^1, T^1 following the same rule,

$$Z_i^1 + T_i^1 = 2 \times T_{i+1}^2 + Z_i^2 \quad (2)$$

The sum S is computed by adding Z^2 and T^2 in stage 3.

Fig. 1: Using stored transfer representation for addition, the rippling of carry from the lower significant digit can be avoided. Two ternary numbers X and Y are added to find their sum S in three stages

III. PROPOSED METHODOLOGY TO IMPLEMENT CARRY-FREE ADDITION IN MEMORY ARRAY

In this section, we give an overview of steps to implement carry-free addition in memory array. The architecture of the proposed in-memory computing system is depicted in Fig. 2. To perform addition, the ternary data stored in the memory array is transferred to the ‘processing array’ which performs addition by processing the data in binary format. As depicted in Fig. 2, addition is performed as a sequence of READ and WRITE operations. Reading involves sensing the stored resistance state, and, in ReRAM technology, sensing ternary data (or differentiating between more than 2 states) is a challenge due to low resistance margin and random variations in the resistive states [16]. Therefore, the ternary data is converted

Fig. 2: Algorithm to perform stored transfer addition of Fig. 1 in memory array. The three stages of Fig.1 are implemented in 11 steps which need 22 memory cycles

to binary data and processed and the final sum is written again to the memory array as ternary data. Both the memory array and the processing array are 1T–1R array configurations¹. We preferred a 1T–1R configuration over a 1S–1R since ternary data (three states) can be stored reliably by varying the gate voltage of the transistor during SET process [17]. Moreover, due to the absence of sneak paths, both writing and reading can be performed energy-efficiently in a 1T–1R array [6].

As depicted in Fig. 2, two ternary numbers can be added in 11 steps. We shall illustrate the 11 steps by considering two ternary numbers: $X = (2120100)$ and $Y = (1222110)$ and elaborate each step (Fig. 3). We have deliberately chosen X and Y such that all the six combinations of trits which occur during addition ($X_i, Y_i = (0,0), (0,1), (1,1), (0,2), (1,2), (2,2)$) are taken into account. In the memory array, the numbers to be added are present as ternary numbers *i.e.*, $X_i, Y_i \in 0,1,2$ and in a resistive RAM, ternary data is encoded as resistance. Trit 0,1,2 are stored as three distinct resistances. First, the ternary data is read out of the memory array by a ternary sense amplifier which can distinguish between the three states. The resistance corresponding to trits 0,1,2 are converted to bits (MSB,LSB) = (0,0),(0,1),(1,1) respectively (Step 1). Then, X

and Y are written to the processing array in Steps 2 and 3. Since each digit of (X, Y) is two bits, the MSB of X is written first and the LSB of X is written in the row below. In this manner, the numbers to be added X and Y are available in binary format (rows 2-5 of Fig. 3). To calculate the sum digit Z_i and transfer T_{i+1} , we employ a READ & COUNT Circuit (RCC). The RCC is basically an array of 3-bit counters, one for each column of the processing array. When a row is activated, the RCC reads the bit in that row and if it is '1', it produces a negative pulse (if '0', no pulse is produced). If rows 2–5 are read consecutively in 4 cycles, $C_2C_1C_0$ will have the number of '1's present in the column (see Fig. 3). A special case occurs when both the inputs are '2' (Refer truth table for stage 1). In this case, T_{i+1} must be '2' and Z_i must '0'. Since we want to process purely in binary, T_{i+1} must be '11'. To accomplish this, the 3-bit counter is designed to count $000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 110$. Consequently, after cycle 9, T^1 will be available at (C_2C_1) and Z^1 will be available at C_0 . In Cycle 10, Z^1 is written to the processing array followed by T^1 in cycle 11 and 12. Note that T^1 is not written exactly below Z^1 , but left-shifted by one position because T^1 is transfer digit. Again, rows 3-5 are read consecutively in 3 cycles to produce Z^2 and T^2 at (C_1C_0) of the RCC. Z^2 and T^2 are written into the array in cycles 16 and 17 followed by read and count at rows 4,5. At the end of 19 cycles, sum S will be available at (C_1C_0) . Writing S to the ternary array requires 3 cycles because each trit is stored as a distinct resistance. Hence to write (021111010) to the ternary array, (0_ _ _ _ _ 0_0) is written in cycle 20 followed by (_ _1111_1_) in cycle 21 followed by (_2_ _ _ _ _ _) in cycle 22.

IV. CIRCUIT-LEVEL IMPLEMENTATION

ReRAM is an emerging technology and devices with diverse properties are being reported *i.e.*, the LRS, HRS, resistance window, threshold voltage at which the device switches vary from device to device. To have a realistic investigation, we considered the Resistive RAM devices manufactured at IHP². The 1T–1R is constituted by NMOS transistor manufactured in IHP's 130 nm CMOS technology, whose drain is connected in series to the RRAM ($TiN/Hf_{1-x}Al_xO_y/Ti/TiN$). The cells have a mean HRS of 133.3 k Ω . During SET process (HRS $\rightarrow$ LRS) transition, the device can be programmed to a LRS of 11 k Ω or 7.5 k Ω with a gate voltage of 1.2 V and 1.6 V, respectively. Hence, the three resistance states are 133 k Ω , 11 k Ω , and 7.5 k Ω and they are used for representing the trits 0, 1, and 2, respectively.

A. Ternary Sense Amplifier

Fig. 4 shows our solution to read trit from the 1T–1R array. As stated, resistance 133 k Ω , 11 k Ω and 7.5 k Ω have to be read as (MSB,LSB) = (0,0), (0,1) and (1,1) respectively. Differentiating between 11 k Ω and 7.5 k Ω needs a robust Sense Amplifier (SA). Furthermore, ReRAMs are prone to

¹In ReRAM, two configurations are adopted for the memory array: 1Transistor–1Resistor (1T–1R) and 1 Selector–1Resistor (1S–1R). 1T–1R uses an access transistor for each ReRAM cell, while 1S–1R uses a selector device to minimize the sneak currents while reading and writing to the array.

²Innovations for High Performance Microelectronics– Leibniz-Institut für innovative Mikroelektronik, Frankfurt Oder, Germany

The diagram illustrates the ternary array architecture for adding two ternary numbers, X and Y. At the top, inputs are shown as ternary strings: $(2120100)_3 = (196)_{10}$ (labeled X) and $(1222110)_3 = (182)_{10}$ (labeled Y). These are converted to bits via a Triter-to-Bit converter, resulting in bit pairs (0,0), (1,0), and (2,1).

The main component is the **PROCESSING ARRAY**, which consists of rows indexed 1 to 5. Row 1 contains the first bits of X and Y. Rows 2 and 3 contain the remaining bits of X and Y. Rows 4 and 5 contain the remaining bits of X and Y. The array also includes intermediate results Z¹, T¹, Z², T², and S.

To the right of the processing array is the **READ & COUNT CIRCUIT**. It takes inputs from the processing array and outputs the final result S. The circuit includes counters for Z¹, T¹, Z², T², and S.

A cycle-by-cycle breakdown is provided on the far right:

- Cycle #1:** Read ternary numbers to be added
- Cycle #2,3:** Write ternary number X to processing array
- Cycle #4,5:** Write ternary number Y to processing array
- Cycle #6-9:** Read and Count
- Cycle #10:** Write Z¹
- Cycle #11,12:** Write T¹
- Cycle #13-15:** Read and Count
- Cycle #16:** Write Z²
- Cycle #17:** Write T²
- Cycle #18,19:** Read and Count
- Cycle #20-22:** Write S to ternary memory

The final output is shown at the bottom: $(021111010) = (378)_{10}$.

The diagram illustrates the 3T1R array architecture. It features a crossbar array of 1T1R units. A read current I_{READ} is applied to the word line (WL) through a sense amplifier (SA). The bit line (BL) is connected to the bit line sense amplifier (BSA). The array is divided into two sections: a current-starved section (left) and a current-saturated section (right). The current-starved section has a current source I_{FF} and a sense amplifier with delay1. The current-saturated section has a current source I_{FF} and a sense amplifier with delay2. The output of the current-starved section is labeled 'LSB' and the output of the current-saturated section is labeled 'MSB'. A table below the diagram shows the mapping of trit values to bit values (MSB, LSB) and the corresponding resistance values (133 kΩ, 11 kΩ, 7.5 kΩ).

	trit	bit (MSB, LSB)
133 kΩ	0	(0, 0)
11 kΩ	1	(0, 1)
7.5 kΩ	2	(1, 1)

The timing diagram below the table shows the sequence of events. The word line (WL) is activated, followed by the bit line (BL) being activated. The current-starved section (left) has a delay1, and the current-saturated section (right) has a delay2. The output of the current-starved section is labeled 'LSB' and the output of the current-saturated section is labeled 'MSB'. The timing diagram shows the sequence of events: EN, delay1, t_{delay1}, delay2, t_{delay2}, and EN.

spatial (device-to-device) and temporal variations (cycle-to-cycle) [16]. Hence, we considered a SA proposed for STT-MRAM which have a narrow margin between its states [18]. The SA proposed in [18] is used to differentiate between two states (binary sensing) and we augmented it with extra circuitry to differentiate between three states (Fig. 4).

to shape the EN signal to have a steep slope when it reaches I_{FF} . Since the edge-triggered flip flops have EN_{delay1} and EN_{delay2} as the clocks, the outputs MSB and LSB of the sense amplifier can sense the ternary state as a binary vector (MSB, LSB) . In time-based sensing [18], V_{BL} controls the arrival of the EN signal at I_{FF} , i.e., a higher resistance results in a large value for V_{BL} and I_{FF} goes low earlier. In this manner, the EN signal is delayed (in proportion to V_{BL}) and will be available at I_{FF} . I_{FF} is fed to two flip-flops, which produces output signals MSB and LSB . Note that the flip-flops are edge triggered by different clock signals EN_{delay1} and EN_{delay2} . (EN_{delay1} and EN_{delay2} are set to different time steps t_{delay1} and $t_{delay1} + t_{delay2}$ by inverter delay chains). As depicted in Fig. 4, for 133 k Ω , I_{FF} is low at the rising edge of EN_{delay1} and EN_{delay2} and consequently sensed as (0,0). But for 11 k Ω , I_{FF} is high at the rising edge of EN_{delay1} and low at the rising edge of EN_{delay2} and hence sensed as (0,1). Thus, according to the stored resistance values 133 k Ω , 11 k Ω , and 7.5 k Ω , the SA produces the output vectors (0,0) for trit 0, (0,1) for trit 1, and (1,1) for trit 2. The ternary SA was designed in IHP's 130 nm CMOS technology and correct sensing of the three states was verified by simulation.

The WRITE circuit is the part of the peripheral circuitry responsible for writing data to the memory and processing array. To minimize latency, the WRITE circuit must have the capability to program multiple cells in parallel. An operational amplifier is used to accomplish this, as depicted in Fig.5. The operational amplifier acts as a voltage regulator and is also able to deliver the required current to program eight 1T-1R cells simultaneously [5]. Writing binary data is straightforward in ReRAM technology – the cell is programmed to LRS (the filament is formed between the electrodes) by applying a positive voltage to the *BL* while *SL* is grounded. The cell

Fig. 5: Write circuit: Op-amp regulates the voltage while driving enough current to write into the cell. In 1T-1R configuration, three distinct resistance can be stored reliably by modulating the gate voltage.

is programmed to HRS (the filament is ruptured) by applying a positive voltage to the SL while BL is grounded. A voltage of opposite polarity is needed across the RRAM cell to break the filament (see Fig. 5). For writing ternary data, we need one more state. This can be accomplished in 1T-1R configuration by varying the gate voltage of the transistor which in turn changes the compliance current during the SET process (HRS $\rightarrow$ LRS). As depicted in Fig. 5, a voltage of 1.2 V at the gate programs the cell to a LRS of 11 kΩ. A higher gate voltage of 1.6 V during the SET process programs the cell to a LRS of 7.5 kΩ. A higher gate voltage results in a higher compliance current and consequently a thicker filament [17]. A thicker filament between top and bottom electrodes forms a wide conductive path, thus lowering the resistance. To verify the WRITE circuit, the 1T-1R cell was modeled by fitting the Stanford-PKU model to characteristics of IHP's RRAM [17]. V_{WRITE} of 1.2 V was used and simultaneous writing of 8 1T-1R cells was verified by simulation.

C. READ and COUNT Circuit (RCC)

Fig. 6: (a) Read and count circuit (b) Illustration of the processing area and its peripheral circuit required for 7-trit (8-bit) addition

The READ and COUNT circuit is the crucial part of the proposed methodology to perform carry-free addition in the memory array. As explained in Section III, the sum digit Z_i and transfer digit T_{i+1} are computed by reading out and counting the number of '1's. As explained in the previous section, '0' is stored as 133 kΩ and '1' is stored as 11 kΩ in the processing array. Since the number of '1's in a column have to be counted, a resistance of 11 kΩ has to be differentiated from 133 kΩ and then counted. In essence, we need a sense amplifier followed by a counter. As depicted in Fig. 6, the Schmitt-Trigger (ST) circuit functions as a sense amplifier. To perform READ and COUNT operation in a column, I_{READ} is injected to the 1T-1R cell as in a memory READ operation. The resistance of the ReRAM cell is transformed to an appropriate voltage ($V_{BL} \approx I_{READ} \times R_{1T-1R}$) and fed to the six-transistor ST circuit which produces a negative pulse if V_{BL} is below a certain threshold voltage. This negative pulse is fed to a three-bit counter which outputs $C_2C_1C_0$ (flip-flops Q_2, Q_1, Q_0 in Fig. 6 are negative edge-triggered). The RCC was designed in IHP's 130 nm CMOS technology and $I_{READ} = 5 \mu A^3$ was used during RCC operation. The ST circuit has an upper threshold voltage (V_{TH}) of 0.7 V and a lower threshold voltage of 0.5 V (V_{TL}) [20]. It must be noted that four consecutive rows must be read and counted to compute Z^1, T^1 (three rows for Z^2, T^2 and two rows for Sum, S). When a '0' is read (133 kΩ), V_{BL} is 0.6 V which is above V_{TL} . Hence the output of ST is held high at V_{DD} . When a '1' is read (11 kΩ), V_{BL} is 0.075 V which is below V_{TL} . Hence the ST outputs goes low. At the end of the READ operation in the first row, corresponding WL is deactivated, and V_{BL} goes high (access-transistor of cell is switched OFF). Consequently, ST output goes high again producing a negative pulse. In this manner, the number of '1's in a column are converted into negative pulses and are counted by the three-bit counter.

V. COMPARISON WITH BINARY IN-MEMORY ADDERS

To the best of our knowledge, this is the first work to propose in-memory carry-free addition methodology. In this section, we compare our work with other in-memory adders which use different logic primitives and adder architectures like parallel-prefix configurations to minimize latency of addition. Table I compares the latency and area/peripheral requirement of our carry-free adder with the best performing binary adders (other binary adders with $O(n)$ latency do exist, but not compared here since their latency ≈ 200 cycles for 32-bit addition, see [5]). As plotted in Fig. 7, carry-free addition outperforms best binary adders for 32-bits and more. The Majority+NOT based parallel-prefix adder [5] is the only binary adder which competes well with the speed of carry-free addition (26 cycles for 32-bit addition) while not requiring huge peripheral modifications. For the 7-trit addition (7-trit is

³ $I_{READ} = 5 \mu A$ is chosen because it forms a small voltage of ≈ 0.6 V across the 1T-1R cell during RCC operation. A larger I_{READ} can be used but can disturb the stored data *i.e.*, accidental WRITE operation which will result in the wrong output at the counter. Minimizing I_{READ} avoids read-disturb event in addition to being energy-efficient.

equivalent to 8-bit), we need a 9×9 processing area and 9 RCC circuit and a WRITE circuit, as illustrated in Fig. 6-(b). In Fig. 6-(b), Z^2, T^2 could have been overwritten on Z^1, T^1 , requiring a lesser area, but this was not pursued due to limited endurance of ReRAM devices *i.e.*, each cell in the 9×9 area is switched once during 7-trit addition. We are not able to compare the energy of adders since the energy depends on the switching energy of ReRAM cell and vary from device to device. The energy of our adder will consist of the energy to READ and WRITE, which can be achieved energy-efficiently in 1T-1R configuration due to the absence of sneak currents [2]. WRITE latency (≈ 50 - 100 ns) is greater than the READ latency (≈ 20 ns) for our adder and this true of all ReRAM-based adders since this is typical of ReRAM technology.

TABLE I: Carry-free adder vs. other binary in-memory adders

Primitive	Adder Type	Latency	Array Area/Periphery/Modification
ORNOR	Parallel-clocking	$2n+15$	n 1S-1R arrays modified with select transistor and R_G [21]
RIMP/NIMP*	Pre-calculation	$2n+4$	Two CRS-arrays with READ/WRITE circuits [22]
XOR+AND+OR	Ripple carry	$2n+2$	$6n+3$; modified array to accommodate a switch for each XOR operation and a resistor for each AND operation [23]
XOR+MAJ	Ripple carry	$2n+2$	$3n$, twin arrays with heavily modified peripheral circuits (row-decoders, Sense amplifiers) [24]
OR+AND	Parallel-prefix	$8\log_2(n)+13$	$(5+\log_2(n)) \times n$ of main array with row/column decoders modified to be able to apply inputs [9]
Majority+NOT	Parallel-prefix	$4\log_2(n)+6$	$6 \times (8n+16)**$ of main array with minor modification to row-decoder [5]
Count	Carry-free	22	One $9 \times (n+1)$ processing array with $(n+1)$ READ and COUNT Circuits (RCC) and a WRITE circuit

*RIMP/NIMP stands reverse implication and inverse implication in a Complementary Resistive Switch (CRS) based adder. **This huge array area requirement is because the authors considered the area of the Sense Amplifier (8 columns share a SAs, pitch-matching) which others works have not considered.

VI. CONCLUSION

In this work, we have proposed, for the first time, a method to implement carry-free addition in the memory array. The carry-free addition was accomplished by STNR as opposed to conventional adders which pursue a binary representation. Carry-free addition could accelerate in-memory computation since they require $O(1)$ latency. The proposed carry-free addition can be reliably performed in 1T-1R Array and this was verified by simulation. The proposed technique incurs a latency of 22 memory cycles, which outperforms all binary in-memory adders for $n \geq 32$. The price one has to pay is increased peripheral hardware since n -bit addition requires a $9 \times (n+1)$ processing array with its peripheral circuitry (WRITE circuit and $n+1$ RCC). The proposed adder will be

Fig. 7: Carry-free addition incurs less latency when compared with other best performing binary adders in memory

energy-efficient since it performs addition by memory READ and WRITE operations and both can be performed without leakage currents (sneak-paths) in 1T-1R configuration.

ACKNOWLEDGMENT

This research was funded by Deutsche Forschungsgemeinschaft (DFG) - Integrierte Memristor-Basierte Rechner-Architekturen (IMBRA) (Project number 389549790).

REFERENCES

- [1] G. Singh *et al.*, “A review of near-memory computing architectures: Opportunities and challenges,” in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 608–617.
- [2] J. Reuben *et al.*, “A taxonomy and evaluation framework for memristive logic,” in *Handbook of Memristor Networks*, L. Chua, G. C. Sirakoulis, and A. Adamatzky, Eds. Cham: Springer International Publishing, 2019, pp. 1065–1099.
- [3] D. Ielmini and H.-S. P. Wong, “In-memory computing with resistive switching devices,” *Nature Electronics*, vol. 1, pp. 333 – 343, 2018.
- [4] J. Reuben *et al.*, “Memristive logic: A framework for evaluation and comparison,” in *Power And Timing Modeling, Optimization and Simulation (PATMOS)*, September 2017, pp. 1–8.
- [5] J. Reuben and S. Pechmann, “Accelerated addition in resistive ram array using parallel-friendly majority gates,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–14, 2021.
- [6] J. Reuben, “Binary addition in resistance switching memory array by sensing majority,” *Micromachines*, vol. 11, no. 5, 2020.
- [7] J. Reuben and S. Pechmann, “A parallel-friendly majority gate to accelerate in-memory computation,” in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 93–100.
- [8] J. Reuben, “Rediscovering majority logic in the post-cmos era: A perspective from in-memory computing,” *Journal of Low Power Electronics and Applications*, vol. 10, no. 3, 2020.
- [9] A. Siemon, S. Menzel, D. Bhattacharjee, R. Waser, A. Chattopadhyay, and E. Linn, “Sklansky tree adder realization in 1s1r resistive switching memory architecture,” *The European Physical Journal Special Topics*, vol. 228, no. 10, pp. 2269–2285, 2019.
- [10] D. Fey, “Using the multi-bit feature of memristors for register files in signed-digit arithmetic units,” *Semiconductor Science and Technology*, vol. 29, no. 10, p. 104008, 2014-09, publisher: IOP Publishing.

- [11] D. Fey, M. Reichenbach, C. Söll, M. Biglari, J. Röber, and R. Weigel, "Using memristor technology for multi-value registers in signed-digit arithmetic circuits," in *Proceedings of the Second International Symposium on Memory Systems - MEMSYS '16*. ACM Press, 2016, pp. 442–454.
- [12] D. Wust, D. Fey, and J. Knödtel, "A programmable ternary CPU using hybrid CMOS/memristor circuits," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 33, no. 4, pp. 387–407, 2018.
- [13] G. Metze and J. E. Robertson, "Elimination of carry propagation in digital computers," in *IFIP Congress*, 1959.
- [14] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, 1961.
- [15] G. Jaberipur, B. Parhami, and M. Ghodsi, "An Efficient Universal Addition Scheme for All Hybrid-Redundant Representations with Weighted Bit-Set Encoding," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 42, no. 2, pp. 149–158, Feb. 2006.
- [16] J. Reuben, M. Biglari, and D. Fey, "Incorporating variability of resistive ram in circuit simulations using the stanford-pku model," *IEEE Transactions on Nanotechnology*, vol. 19, pp. 508–518, 2020.
- [17] J. Reuben, D. Fey, and C. Wenger, "A modeling methodology for resistive ram based on stanford-pku model with extended multilevel capability," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 647–656, 2019.
- [18] Q. Trinh, S. Ruocco, and M. Alioto, "Time-based sensing for reference-less and robust read in stt-mram memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 10, pp. 3338–3348, 2018.
- [19] D. Fey and J. Reuben, "Direct state transfer in mlc based memristive rram devices for ternary computing," in *2020 European Conference on Circuit Theory and Design (ECCTD)*, 2020, pp. 1–5.
- [20] J. Reuben and D. Fey, "A time-based sensing scheme for multi-level cell (mlc) resistive ram," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2019, pp. 1–6.
- [21] A. Siemon *et al.*, "Stateful three-input logic with memristive switches," *Scientific Reports*, vol. 9, no. 1, p. 14618, 2019.
- [22] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 1, pp. 64–74, 2015.
- [23] N. TaheriNejad, "Sixor: Single-cycle in-memristor xor," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 5, pp. 925–935, 2021.
- [24] F. Pinto and I. Vourkas, "Robust circuit and system design for general-purpose computational resistive memories," *Electronics*, vol. 10, no. 9, 2021.