Towards In-Memory Computing: Arithmetic Operations on Real Memristors

Thore Kolms
Universität zu Lübeck
Lübeck, Germany
thore.kolms@student.uni-luebeck.de

Christine Lang
Universität zu Lübeck
Lübeck, Germany
christine.lang@student.uni-luebeck.de

Andreas Waldner
Universität zu Lübeck
Lübeck, Germany
andreas.waldner@student.uni-luebeck.de

Philipp Grothe
Institute of Computer Engineering
Universität zu Lübeck
Lübeck, Germany
grothe@iti.uni-luebeck.de

Jan Haase
Department of Computer Science
Nordakademie
Elmshorn, Germany
jan.haase@nordakademie.de

***Abstract*—In-memory-computing is an emerging approach that aims to shift computational load away from CPUs. It does so by taking over simple calculations that can be performed in memory. Apart from improving performance for these operations, this also results in a lower energy consumption, effectively rendering this technique very suitable for embedded systems.**

Due to being variable as well as non-volatile resistors, memristors are capable of storing analog values. This renders them particularly useful for in-memory computing.

This paper presents a prototypical implementation of analog calculations (addition, subtraction and multiplication), including a way of representing the value zero. The prototype shown is based on an ESP32 microcontroller and typical calculations currently take around 1μs.

***Index Terms*—Memristor, In-Memory Computing, Multi-Level-Cell, Analog Computing, Basic Arithmetics**

I. Introduction

In-memory-computing (IMC) is a method where computation is shifted from the CPU to the memory. Due to this shift, data does not have to be routed from memory to the CPU to perform operations and the results do not have to be written back to memory after the calculation. Any arithmetic operation can be performed directly in memory. This relieves the bus, since values do not have to be sent back and forth between memory and arithmetic units permanently. IMC also improves data access times and, hence, the speed of operations.

For reasons of symmetry, Leon Chua stated in 1971 in his work [1] that a fourth fundamental passive component besides the resistor, capacitor and inductor has to exist. This component then fills the connection between electric flux and charge. Moreover, the underlying assumption led to the device now known as "memristor". This term is a neologism created from the combination of the words memory and resistor. Thus, the memristor is an electrical resistor the resistance of which can be changed by a current flow. Depending on the direction of the current, the resistance of the memristor can be increased or decreased. If there is no voltage on the memristor, it maintains its resistance. In 2008, Strukov, Snider, Stewart and Williams succeeded in developing the first functioning memristor for HP Labs [2]. In contrast to conventional storage media, which have a binary storage capacity, memristors can store several bits in one component on the basis of the different resistance values. This not only means that memristors are theoretically more energy and space efficient for storage than conventional memories, but also offer the possibility of storing analog values. It is, therefore, conceivable that a memristor could be used for IMC.

In order to be useful as memory, memristors need to be able to retain several bits of data through multi-level cells [3], [4]. This means that, for each memristor, certain resistance intervals correspond to specific values. Allowing analog memory, memristors can also be used to facilitate information processing through neural networks as part of developing artificial intelligence [5], [3], [6]. Additionally, as [7] points out, memristors operate in a very similar way to the synapses connecting neurons in the human brain. Thus, as demonstrated by [5], memristors can also be utilized for replicating biological learning processes in neuromorphic computing, such as spiking neural networks. As these types of networks tend to consume large amounts of power and space, memristors are very promising due to their storage potential and low energy usage [5], [3]. Moreover, the possibility of performing in-memory computing would render the necessary calculations much more efficient [8].

In [9], possible circuits for performing arithmetic operations as well as a circuit for setting memristors are presented. The latter is fairly similar to the circuits suggested in [10] and [11] but remains rather abstract. In other words, the proposed circuit is based on ideal components that do not actually exist, such as constant current sources or a function for calculating absolute values. Also, it does not employ voltage levels or charges to represent operands. Instead, memristances are utilized.

None of the findings outlined above focuses on the realization of arithmetic operations in hardware. They rather

constitute theoretical approaches and simulations. Thus, the suggested hardware implementations of arithmetic operations have, in part, been demonstrated to be fully functional in simulations. They have, however, not been physically built and tested. The same applies to in-memory computing architectures for matrices, most of which have only been conceptualized in theory. As a result, there is no product that can perform arithmetic or matrix operations for in-memory computing using memristors as of yet. A practical implementation using real memristors is, therefore, presented in this paper.

The following Sections deal with the methodology of setting memristors, arithmetic operations, and overflow handling. Afterwards the results and a conclusion with an outlook are presented.

II. Methodology

This Section will deal with precisely tuning a memristor to a specific value and the execution of basic arithmetic operations.

A. Setting Memristance and Writing Results

In order to be able to calculate with memristors as precisely as possible, there has to be a way of setting the memristor to a certain value with high precision in a very short time.

Other works such as [11] have already developed circuits with different approaches for setting the memristance. In theory, this circuit works well, but before it can start setting the memristor it has to read its value to determine in which direction it has to be changed. When this circuit is modified so it can set memristors in either direction via a feedback loop, it starts to oscillate once it reaches the target value as seen in Figure 1. Such a settlement inaccuracy is detrimental to the use of in-memory computing. Since this work does not exclusively deal with the precise setting of the memristance, but needs the setting for further development, the process was realized with multiple pulses. This puts a higher load on the microcontroller which leaves less computing power for other tasks. Furthermore, the speed of the whole system suffers from this.

After the arithmetic operations, the results need to be saved. Setting memristors with pulses that are managed by a microcontroller does not work very well for writing analog results into memristors. Another strategy for saving them has to be employed. One such option consists of an analog solution, such as the one demonstrated in [11]. This circuit then works with a reference voltage and utilizes a feedback loop. The same circuit was considered for initial setting of the memristors but is even more suitable for this situation. Hybrid solutions are possible [12].

At the moment, memristor cells are still fairly expensive and not easily obtainable. For this reason, architectures of memristor-based computing systems should employ them sparingly. An operational amplifier currently costs very little. Thus, it is usually more economical to design more complex external circuits if the number of memristors can be reduced this way. Similarly, analog switches or multiplexers tend to be rather inexpensive and should be included before additional memristors are added to the architecture.

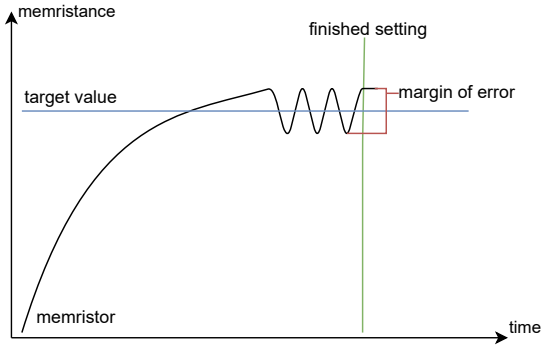


Fig. 1: Typical change of the memristance in the process of setting a memristor. Overshoot and oscillation around the target value can be seen. The margin of error is determined by the rate of change and the time it takes to counteract. The actual error depends on the time when the setting procedure is stopped but can be described as random within this margin when no further control mechanisms are applied.

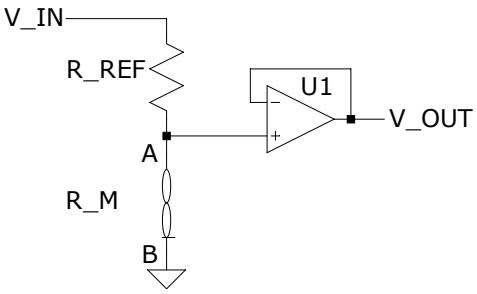


Fig. 2: A memristor memory cell. The memristor is part of a voltage divider, the output of which is additionally buffered. This way the output voltage V_{out} is given by the voltage divider equation $V_{\text{out}} = V_{\text{in}} \cdot \frac{R_{\text{REF}}}{R_{\text{REF}}+R_{\text{M}}}$ and independent from any additional load.

Nevertheless, the number of components should always be kept at a minimum. In this context, it is essential to evaluate the necessity of functions that can only be realized in specific architectures, such as parallel memory access or parallel usage of different arithmetic units.

These circuits were already established in earlier research but have since been refined and the implementations carried into the next stage to prototyping to improve results [13].

B. Arithmetic Operations

For the implementation of arithmetic operations using memristors two approaches are possible: On one hand, the resistances constituting the stored values can be explicitly used for the calculation. For example, the addition of two resistances can be achieved by simply connecting them in series. On the other hand, voltages controlled by the resistance of the memory cell can be used to implicitly implement the operations on memristive values. Such a memory cell is shown in Figure

2. As long as the function between resistance and voltage is known, the same results are possible. For not all arithmetic operations are easily achievable using explicit resistances, the approach of using relating voltages was chosen. Each memristor is paired with a resistor of fixed value to constitute a voltage divider.

Addition is achieved by the use of a summing amplifier as a voltage adder. The input voltages must be buffered to prevent unwanted interaction between the resistor values of the memory cells. The circuit is shown in Fig. 3a.

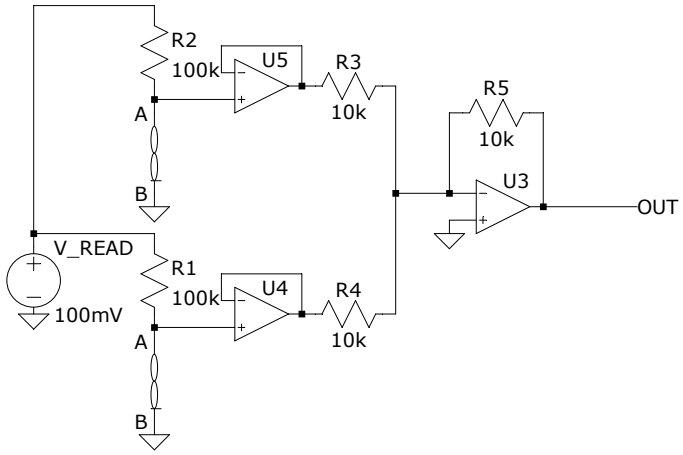
Subtraction is implemented the same way as addition, as it is practically the same operation with one negative input. The aforementioned buffer can be modified to simultaneously invert the voltage, as can be seen in Fig. 3b.

Multiplication needs four operational amplifiers to multiply two input voltages. This is done by first calculating their logarithms, using two amplifiers in a closed loop configuration directionally constrained by a diode, then summing according to the addition configuration and finally calculating the exponential of the sum through another closed loop amplifier. This is shown in Fig. 3c.

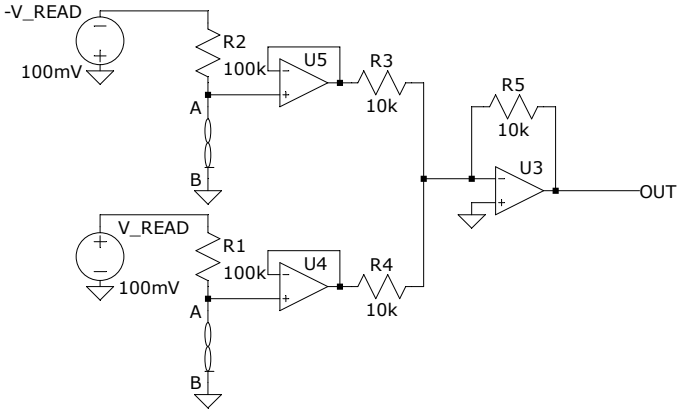
Division poses the greatest challenge of all operations. There are existing integrated circuits for performing division like the Analog Devices AD538. The circuit used for the division in the AD538 is very similar to the one for the multiplication utilized in this work. It can be used for future improvements. However, the majority of use cases for in-memory computing nowadays focus on MAC operations (multiply and accumulate), which relegates the implementation of division to a secondary role.

C. Overflow Handling

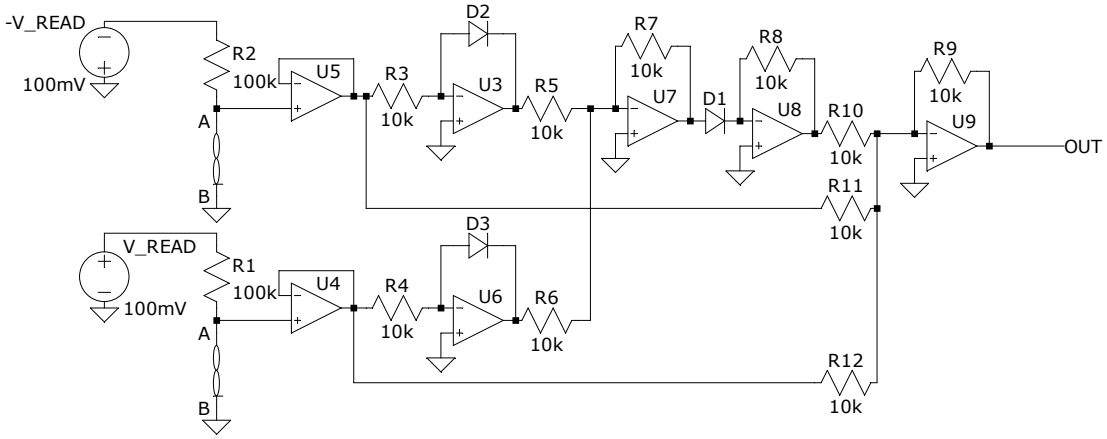
Two memristors representing their maximum possible values when added or multiplied will consequently result in a value greater than the maximum value that can be stored in a single memristor. Calculations utilizing voltages instead



(a) Addition



(b) Subtraction



(c) Multiplication

Fig. 3: Shown are circuits implementing the addition, subtraction and multiplication of two voltages controlled by resistive memory cells. The memory cells are still present in the circuit diagrams with V_{out}, according to Figure 2, being located at the junctions after U4 and U5 respectively.

of explicit resistances could, depending on the implemented circuit, result in either lost information or even voltages unsuitable for the use of memristors. In conventional arithmetic units these cases can be mitigated by an additional carry bit. Analogously, the value can be stored in multiple memristors. The circuit for writing the result into memristors must be adapted accordingly and the number of necessary memristors depends on this implementation.
Additionally, the highest possible value stored in a memristor corresponds to the highest possible voltage in the circuit to guarantee the best possible resolution. An overflow would therefore exceed this voltage. Especially with the use of operational amplifiers this would cause the supply voltages to be insufficient. This is handled by a fixed reduction within the stage of calculation. Integrating the reduction into this step ensures that an excessive voltage is never needed and the loss of precision is minimized. The reduction must be calculated according to the worst case of each operation, resulting in a reduction of factor n for the addition of n values and a reduction of factor $n-1 \cdot V_{\text{max}}$ with V_{max} denoting the maximum value of a single memory cell for the multiplication. This causes the loss of bijectivity between voltage and value which must be again accounted for in the design of a write circuit.

D. Zero Handling

Another concern is the representation of the value of zero as one or both of the input arguments. In the configuration shown in Fig. 2, the resistance of the memristor would have to be 0Ω to generate a voltage of 0V. Not only is this not possible for common resistors, the resulting lack voltage dropping over the memristor would result in a stuck-on condition for the affected memristor [14]. Additionally, the current increases with decreasing resistance, potentially to harmful levels.

A different value must, therefore, be defined to represent the value of zero. Intuitively, this offset can be defined by the voltage generated by the memory cell when the memristor is at its lowest safely achievable resistance. All represented values are relative to this offset. Choosing an offset higher than the lowest achievable voltage would theoretically allow for simple storage of negative values, as they would still be represented by positive voltage simply defined as negative values. This, however, would require further handling of edge cases to prevent data loss due to underflow. For this reason, it shall be reserved for future research.

Since the actual compute unit, as seen in Fig. 3, expects a voltage of 0V as zero to achieve correct results, the offset must be subtracted before doing the calculation. This can be realized by an analog subtraction circuit, similar to the one in Fig. 3b but with the offset voltage as a fixed input.

Despite eliminating the offset beforehand, an offset must be added back after the calculation. This is primarily due to a loss of precision of the utilized ADC in the range of very low voltages, especially below 50 mV. Furthermore, this way the resulting voltage is potentially compatible to be used as a voltage to set another memristor to explicitly write the result into a memory cell. This is, however, hindered when

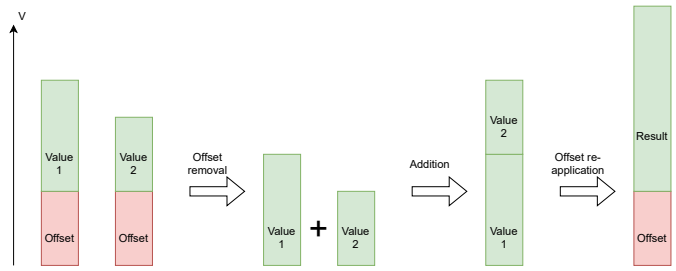


Fig. 4: Visualization of applied offsets in memory cell, while executing the calculation, in this case addition, and reading or storing the result. Before the actual calculation, the offset is removed to gain results without the offset. Its is then added on again. By this procedure, the offset can be handled the same way no matter the type of calculation. The relation of offset size and values is not necessarily representative of the real application.

the scaling described in Subsection II-C is applied, as output voltages no longer correspond to the same resistance in a memory cell as an equivalent input voltage. If scaling is not applied or the effect adequately considered, this would enable the execution of the calculation fully in memory. The general process of adding and subtracting offsets during a whole calculation is shown in Fig. 4.

III. Results

In this Section, the experimental results of a prototype (shown in Fig. 5) system are presented.

A. Precision

The voltages were measured with the internal ADC of an ESP32 which has a resolution of $12Bit$ and a reference voltage of $1110mV$. For the memristor memory cells a Knowm tier-1 BS-AF-W memristor was used in a voltage divider configuration with a $100k\Omega$ resistor. The threshold of these memristors is typically between $150mV$ and $300mV$, so a read voltage of $110mV$ was chosen. In this configuration, the memristor

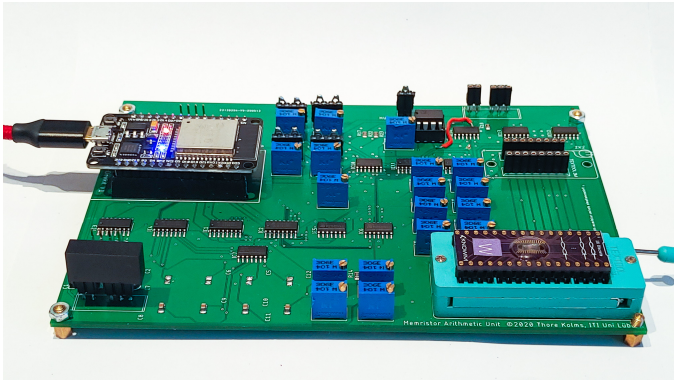


Fig. 5: Implemented prototype on printed circuit board with socketed microcontroller and socketed memristor array. Spindle trimmers allow for adjustments of all critical voltages.

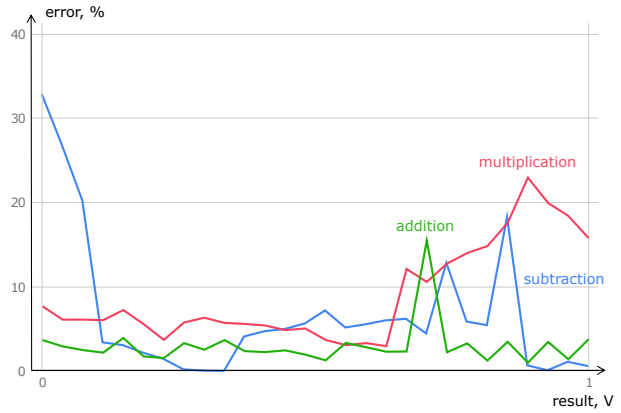


Fig. 6: Measurements to find the best area of operation. As can be seen, values closer to the extremes of the operation parameters will yield inferior results. Measurements were taken after the amplification of the calculation results.

cells have an output range of $50mV$ which has a positive offset of $30mV$ to $50mV$. The output range depends on the specific memristor and on the fixed resistor. The differences in memristors can be compensated by using resistors that are matched to the R_{ON} and R_{OFF} of the memristor in the voltage dividers. The output voltage range is only a tenth of the ADC reference voltage. So a 10x voltage amplifier was used to read the value of the memory cells. After the addition engine a 5x amplifier was used because the output of the addition can be twice as high as the output of a single memristor memory cell. The result tolerances of the addition are typically below 5 %. The precision of multiplication and subtraction vary greatly within the spectrum of possible voltages, as can be seen in Figure 6. Tolerances were calculated from the values that were read from both memristor memory cells and the adder engine with the ESP32. The accuracy of the computation results is allowing about 3.5 bits of precision. The operational amplifiers used are LM324.

B. Timing

Timing wise there are two factors to consider: The time to set the two memristors and the time to calculate the result.

Memristor memory cells have a nonlinear set curve. Changing a value in the upper value range does not take as long as changing the value of the memory cell in the lower voltage range. Thus, the time needed to set the memristor strongly depends on the current resistance of the memristor as well as the target value for the memristor to reach. With regard to the Knowm Wolfram memristors used, it ranges from a few nanoseconds to several hundred milliseconds.

The time to compute the result itself is mainly dependant on the actual implementation. In the used prototype, the computation needs

$$t_{\text{computation}} = t_{\text{multiplex}} + t_{\text{arithmetic}} + t_{\text{amplifier}} + t_{\text{buffer}} \quad (1)$$

where $t_{\text{multiplex}}$ is the time that the multiplexers need to connect the operand memristor memory cells to the correct voltage and also to the arithmetic unit. Even though there are several multiplexers on the signal path, the necessary time only needs to be taken into account once as all of them can switch simultaneously. $t_{\text{arithmetic}}$ denotes the time that the operational amplifiers in the arithmetic unit take to get the correct result value. In between the memristor memory cells and the ALU, the signal is buffered once in order to prevent any corruption of the read value by putting load on the voltage divider. This buffer consists of an operational amplifier in impedance converter configuration and takes time t_{buffer}. In addition, time $t_{\text{amplifier}}$ is needed in order to scale the output of the ALU to the voltage range of the analog-digital converter.

The transition time of the multiplexers used (DG408) is always below $250ns$. The slew rate of the operational amplifier used (LM324) is $0.5V/\mu s$ while the output range of the arithmetic unit is $0V$ to $0.2V$. Taking this voltage and the slew rate, the time $t_{\text{arithmetic}}$ is calculated from

$$t_{\text{arithmetic}_{\text{addition/subtraction}}} = \frac{0.2V}{0.5V/\mu s} = 0.4\mu s = 400ns. \quad (2)$$

The maximum voltage output of the amplifier is $1V$. This means that $t_{\text{amplifier}}$ can be no higher than

$$t_{\text{amplifier}} = \frac{1V}{0.5V/\mu s} = 2\mu s = 2000ns. \quad (3)$$

The buffer located right behind the memory cell has a maximum voltage output of $100mV$. Thus,

$$t_{\text{buffer}} = \frac{0.1V}{0.5V/\mu s} = 0.2\mu s = 200ns. \quad (4)$$

Inserting this value in the equation 1, we get the following upper time barrier for the addition of values in memristor memory cells:

$$\begin{aligned} t_{\text{addition}} &\leq 250ns + 400ns + 2000ns + 200ns \\ &= 2850ns \end{aligned} \quad (5)$$

The slew rate of the multiplier is $20V/\mu s$ and its maximum voltage output lies at $1V$. Therefore, the multiplier needs at most

$$t_{\text{arithmetic}_{\text{multiplication}}} = \frac{1V}{20V/\mu s} = 0.05\mu s = 50ns. \quad (6)$$

Instead of a buffer, a preamplifier based on a LM324 with an amplification factor of 10 is used for the multiplication. Taking the slew rate of the LM324 into account, t_{buffer} is

$$t_{\text{buffer}} = \frac{1V}{0.5V/\mu s} = 2\mu s. \quad (7)$$

The measuring amplifier allows for a maximum voltage level of $1V$, the same as for the addition. As a result, we get a timing of $2\mu s$ here as well.

Substituting these values into equation 1, the time barrier for performing a multiplication is as follows:

$$\begin{aligned} t_{\text{multiplication}} &\leq 250ns + 50ns + 2000ns + 2000ns \\ &= 4300ns \end{aligned} \quad (8)$$

In order to take real timing measurements, the time for the actual calculation must be isolated. This was done by substituting the memristors with fixed resistors and deliberately increasing the voltage applied to one of them. For the circuits doing the calculation, this is equivalent to a change in memristance and the change of the output values can be observed.

In Figure 7 this procedure is shown for the addition. The yellow plot shows the input value changing. When it reaches its target value, the measurement is started to only measure the time needed for the calculation. It is finished when the green plot of the output value settles on a stable level. With regard to the time, this change took $2.4\mu s$. However, it has to be taken into account that this synthetic jump of input values is much bigger than any jump expected in real operating conditions to set an absolute worst case boundary. For the multiplication, this synthetic upper boundary was determined to be around $6\mu s$. While the results of the addition were well reproducible, the susceptibility to noise of the multiplication increased the variance of the measurement, sometimes reaching $8\mu s$. On the ESP32, the computations themselves would take just about a thirtieth of the time but the data would have to be transported from the memory to the ALU over the bus first. This process puts load on the bus and, therefore, leaves fewer resources for other operations. In addition, the increased power consumption as well as the excess heat produced constitute undesirable attributes in any computing system and in embedded systems in particular.

Memristive memory is non-volatile, which makes it a potential candidate for replacing HDDs and SSDs. This would render the loading of values from slow flash memory into main memory redundant. As a result, this would flatten the storage hierarchy and increase the system's speed as data from the lowest storage level can be accessed much faster. Furthermore, accessing data on an HDD takes several milliseconds whereas accessing data in analog memristive memory takes less than a few microseconds using this prototype. Computations on data residing in the storage and not in the memory also take much longer in general.

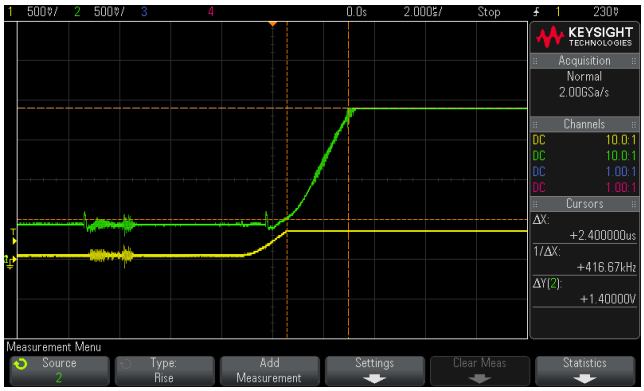


Fig. 7: After changing one input value (yellow plot) for the calculation, the output values adapt accordingly (green plot).

IV. Conclusion

This work shows that, in practice, it is possible to implement arithmetic operations for in-memory computing on real memristors. The precision of 3.5 bits is less than a typical digital ALU, but it has to be kept in mind that these results were gathered from a prototype. More sophisticated implementations such as ASICs of the same circuit would most likely reach higher precision. Also, memristors are in an early phase of development and do have some issues regarding their reliability. Even though the timing is not as fast as a CMOS ALU, memristive memory is non-volatile and, thus, can be used as storage with in-memory-computing capabilities.

Next steps in the project will be measuring the energy consumption and additional timing analysis, as well as subsequent improvements regarding the speed of operation.

References

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

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[3] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1t1r arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, no. 36, Aug. 2016.

[4] L. Wu, H. Liu, J. Li, S. Wang, and X. Wang, "A multi-level memristor based on al-doped hfo_2 thin film," *Nanoscale Research Letters*, vol. 14, no. 177, May 2019.

[5] K. Long and X. Zhang, "Memristive-synapse spiking neural networks based on single-electron transistors," *Journal of Computational Electronics*, vol. 19, pp. 435–450, Dec. 2019.

[6] L.-G. Wang, W. Zhang, Y. Chen, Y.-Q. Cao, A.-D. Li, and D. Wu, "Synaptic plasticity and learning behaviors mimicked in single inorganic synapses of $pt/hfo_x/zno_x/tin$ memristive system," *Nanoscale Research Letters*, vol. 12, no. 65, Jan. 2017.

[7] N. Ilyas, D. Li, C. Li, X. Jiang, Y. Jiang, and W. Li, "Analog switching and artificial synaptic behavior of ag/sio_x : $ag/tio_x/p^{++} - si$ memristor device," *Nanoscale Research Letters*, vol. 15, no. 30, Jan. 2020.

[8] L. Song, Y. Wu, X. Qian, H. Li, and Y. Chen, "Rebnn: in-situ acceleration of binarized neural networks in reram using complementary resistive cell," *CCF Transactions on High Performance Computing*, vol. 1, pp. 196–208, Oct. 2019.

[9] F. Merrikh-Bayat and S. B. Shouraki, "Memristor-based circuits performing basic arithmetic operations," in *Procedia Computer Science*, vol. 3, 2010, pp. 128–132.

[10] O. A. Olumodeji and M. Gottardi, "A pulse-based memristor programming circuit," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[11] Y. Zhao, B. Li, and G. Shi, "A current-feedback method for programming memristor array in bidirectional associative memory," in *2017 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, 2017, pp. 747–751.

[12] P. Grothe and J. Haase, "Memristors for programmable circuits controlled by embedded systems," in *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 37–40. [Online]. Available: https://doi.org/10.1145/3323439.3323985

[13] T. Kolms, A. Waldner, C. Lang, P. Grothe, and J. Haase, "Analog implementation of arithmetic operations on real memristors," in *Proceedings of the 23nd International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 54–57. [Online]. Available: https://doi.org/10.1145/3378678.3391883

[14] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-faults in memristor crossbar arrays using matrix transformations," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 438–443.