# Ternary Arithmetic Pipeline Architectures using multi-bit Memristors

Dietmar Fey

Friedrich-Alexander-University Erlangen-Nürnberg (FAU)
Department Computer Science 3, Chair for Computer Architecture
Email: dietmar.fey@fau.de

***Abstract*—The paper addresses the possibilities exploiting the multi-bit storing features of memristor cells for balanced ternary signed-digit (SD) arithmetic pipelines realising addition, subtraction and multiplication. Different pipeline schemes using memristors as pipeline registers are shown in overview. Their functionality was verified on digital level by SystemC simulations and their performance is comparatively evaluated using analytic methods. In a ternary SD number system the digits of the operands can be assigned not only to 0 and 1 but also to minus 1. Such an SD representation has the great advantage that additions and subtractions can be carried out in $O(1)$ step, i.e., the run time is independent from the word length $N$. However, this requires a physical memory cell that allows to store reliably not only two but at least three states and in addition such a device has to offer fast access times and must be compatible with CMOS logic. All these constraints are fulfilled by memristors. Using memristor based pipeline registers offers different alternatives for implement fast arithmetic architecture circuits that differ from conventional ones. In particular, they allow to use more homogeneous pipelines for realising addition, subtraction and multiplication than current superscalar pipelines, which use for different operations also different pipeline paths. However, for the realisation of a homogeneous pipeline using memristors there are different possibilities. Those were evaluated in this paper concerning latency and hardware effort measured in number of required logic elements for computing and memristors for storing.**

***Keywords–Memristive computing; Ternary logic, Signed-digit arithmetic.***

## I. INTRODUCTION

Since the invention of a nano-device in the HP labs in 2008 by a team around Williams [1] that shows the specific behaviour prognosticated by Leon Chua in a paper of 1971 [2], and which was called by Leon Chua *memristor*, a lot of research launched in the community to use such new devices for digital and analogue logic operations. A memristor is an electronic device that can sustainably change its internal electrical resistance due to a change of an applied magnetic flux generated by electronic charge. Normally, this change of the magnetic flux and charge is generated indirectly by applying a voltage at the poles of a two-terminal memristor device to produce a current flow through it, if the memristor is connected to a circuit. Chua claimed, that if the current-voltage curve of such a device shows hysteresis behaviour and runs through the origin (see Figure 4), then it is a memristor ("if it's pinched, then it's a memristor"). Depending on the direction of the current the changing of the device's resistance can be made reversible. For example, it is possible to mimic the strength of a synaptic interconnect that is controlled by the time difference of two currents arriving at the two poles of the device to realise analogue neuromorphic processing based on the spike time dependent plasticity (STDP) model [3]. Furthermore, complete memristor networks were proposed for digital processing. In such networks targeted changing of the memristors' resistance, called *memristance*, was induced in specific memristors to realise fundamental digital Boolean logic elements [4][5][6].

In some works, the research was also focused on the multi-bit storing feature of a memristor, i.e., the capability to store different resistance levels produced by periodic voltage excitations at the memristor poles. That multi-bit feature is also the topic in this paper. The idea, that was first published in [8], is to use memristors as elementary ternary storing cells for registers in a ternary arithmetic unit. This unit is based on a number system, in which the addition is accelerated by using a balanced ternary representation, i.e., each digit can be assigned to $1, 0$, and $-1$ (in the following denoted as $\bar{1}$).

Based on that addition, further arithmetic circuits like a multiplication can be built up in a pipeline structure. Memristors are to use as multi-bit pipeline registers to store the ternary digits, called *trits*. There are different possibilities for the exact structure of such a pipeline, e.g., to process directly two numbers in signed-digit (SD) representation or a hybrid arithmetic structures operating an SD number and a binary number. All theses alternatives cause different realisation efforts on the logic side, i.e., the number of used gates, and on the storage side, i.e., the number of necessary memristors and transistors for the interface circuitry to convert trits, stored in a single memristor cell, to bits and vice versa. In this paper, some of the different alternatives are shown and are evaluated among each other to select the best for a future memristor-based ternary computer.

The rest of the paper is structured as follows. In Section II, the fundamentals of a memristor and the possibility to store multi-bits in such a single physical storage cell are explained. In Section III, the principal Boolean equations and benefits of SD arithmetic are illustrated. In particular, it is shown why a ternary arithmetic unit is to favour. Afterwards in Section IV, computer arithmetic and memristor technology are brought together by presenting different solutions for arithmetic pipelines using memristor-based registers. These different proposals are evaluated in Section V, in terms of latency, bandwidth and hardware effort. Finally, Section VI summarizes the most important statements of the paper.

## II. MEMRISTOR

The core of a memristor is a so-called transfer oxidation metal, like e.g., $TiO_2$. This metal oxide layer is sandwiched with a ionized layer of the metal oxide, $TiO_{2-x}$, between two metal plates (see Figure 1). Inside this two-terminal

 ISBN: 978-1-61208-389-6

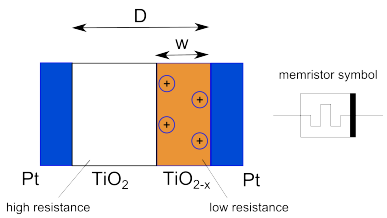


Figure 1. Structure of a two-terminal memristor cell

device electro-chemical processes of oxidation and reduction are taking place to move ions along the interface of the metal oxide and the ionized layer. This moving can be caused by an electrical field applied at the two metal poles. The change of voltage, $V(t)$, and current, $I(t)$, over time generates a magnetic flux, $\varphi$, and a charge, $q$, which both determine the memristance $M$ of the device according to Leon Chua's assumption, expressed in (1) and (2).

$$d\varphi = M dq \tag{1}$$

$$M(q(t)) = \frac{d\varphi/dt}{dq/dt} = \frac{V(t)}{I(t)} \tag{2}$$

This ion transfer is responsible for the sustainable changing of the resistive features of such an element. The electron depleted ionized layer ($TiO_{2-x}$) has a much lower resistance than the metal oxide layer. Both layers determine with different weights according to their lengths the resistance of the whole device (3). $R_{OFF}$ is the resistance if the device is switched off, i.e., no ionized layer is available. $R_{ON}$ holds for the contrast, i.e., the device contains of a complete ionized layer. Since the depletion zone can be shifted in both directions according to the direction, where the voltage is applied, the changing of the resistance is reversible. Therefore it can be memorized and this kind of reversible resistance with memory effect is called *memristance*.

$$R_{MEM}(x) = R_{ON} \cdot x + R_{OFF} \cdot (1 - x), \quad where \;\; x = \frac{w}{D} \;\epsilon\; (0, 1) \tag{3}$$

The ratio of the length of ionized layer, $w$, to the total length of the two layers, $D$, is called the state variable $x$. The changing of this state variable in dependence on the time is expressed in (4),

$$\frac{dx}{dt} = \frac{\eta \cdot \mu_D \cdot R_{ON}}{D^2}\, I(t), \tag{4}$$

where $\mu_D$ corresponds to the drift mobility of the ions, $\eta$ is equal to 1 or $-1$, resp. The last parameter expresses the direction of the applied voltage causing either an expanding or a shrinking of the depletion zone.

The most elaborate presentation of the mathematical derivation of the behaviour of memristors based on ion transfer can be found in [11]. The paper derives a discretization of the differential equation (4), which is shown here by (5) to (8). These equations allow to calculate the changing of the parameters between two discrete time steps $t_i$ and $t_{i+1}$.

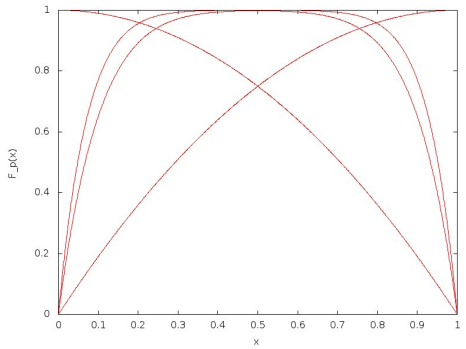


Figure 2. Modelling non-linearity with Biolek's window function for different parameters $p = 1, 5$, and 7. Considered is also $I(t) < 0$, the curve starting at $F_p(0) = 1$.

$$M(x(t_i)) = R_{ON} \cdot (x(t_i)) + R_{OFF} \cdot (1 - x(t_i)), \tag{5}$$

$$I(t_{i+1}) = \frac{V(t_{i+1})}{M(x(t_{i+1}))}, \tag{6}$$

$$x(t_{i+1})/dt = \frac{\eta \cdot \mu_D \cdot R_{ON}}{D^2}\, I(t_i) \cdot F_p(x(t_i)), \tag{7}$$

$$x(t_{i+1}) = x(t_{i+1})/dt \cdot [t_{i+1} - t_i] + x(t_i) \tag{8}$$

The comparatively simple mathematical modelling assumes a linear drifting of the ions according to (4). However, real memristors show a different drift behaviour the nearer the ions are located at the boundaries of the device, where the drifting is slower. This non-linearity is modelled by a so-called window function $F_p$ in (7). Different window functions are proposed in literature [9][10]. In this paper, the window function from Biolek [10], see (9), was selected since it simplifies convergence issues at the device boundaries compared to other window functions. Figure 2 shows the graph for Biolek's window function.

$$F_p(x) = 1 - [x - u(-I)]^{2p}, \quad where \;\; u(I) = \begin{cases} 1, & if I \geq 0 \\ 0, & if I < 0 \end{cases} \tag{9}$$

To investigate the memristor's behaviour more detailed in this paper a numerical simulation program was written in C to implement (5) to (9). The outputs of the simulation program are shown in the following figures. Figure 3 shows the applied input voltage over time. First two positive voltage oscillations are applied at the two terminals of the memristor device. Then, this process is reversed by changing the polarization. Figure 4 shows the I-U behaviour. The typical hysteresis loops for memristors are arising. After each voltage oscillation the level of memristance is changed and another hysteresis loop is valid.

Figure 5 displays the value of the state variable $x$. The multi-state or multi-bit feature of a memristor can be clearly detected. With each excited voltage oscillation the state variable $x$ was shifted to another level. Even if the values are not completely constant and the transition between the levels is smooth each level corresponds to some amount to a discrete memristance level. The level starts not at zero, the initial value

 ISBN: 978-1-61208-389-6

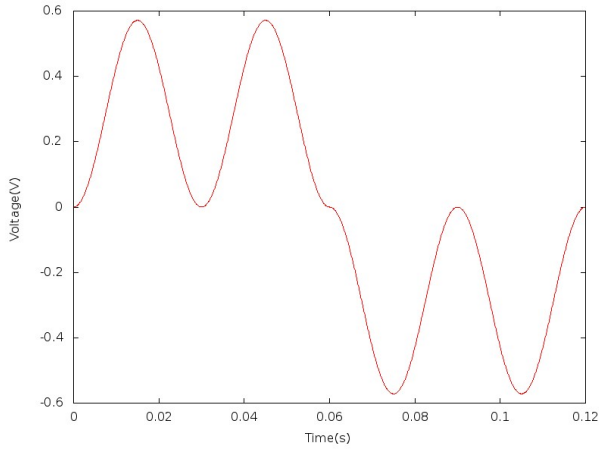


Figure 3. Input excitation with two voltage oscillations

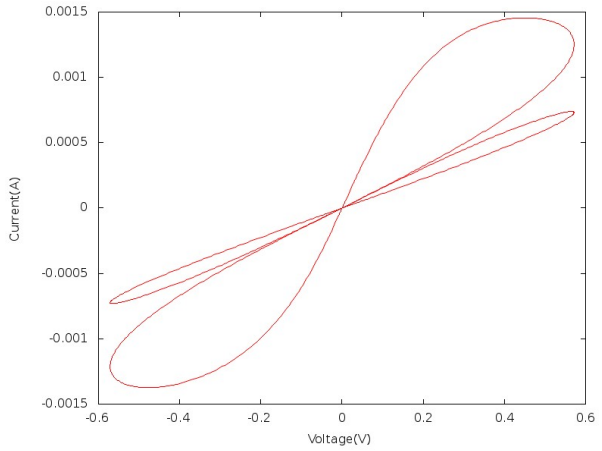


Figure 4. The memristor hysteresis loop, generated for $D = 41nm$, $x(0) = 0.11$, and $p = 7$. For each excited voltage oscillation another hysteresis behaviour is generated

for $x$ was assumed as $0.11$. It expresses an arbitrary assumed ratio between doped and undoped zone at the beginning determined by the memristor's manufacturing process. Further, two levels were injected and then this process was withdrawn by the applied inverted voltage levels. Each of these three levels corresponds to one of the three signed digits, 1, 0, and $\bar{1}$, which are stored in one nano-sized physical cell. Such a cell is the base for pipeline registers used in the proposed SD arithmetic circuits proposed in the next section.

## III. SD-ARITHMETIC

As already mentioned in the introduction the great advantage of arithmetic circuits using SD numbers is that the execution time of an addition has a complexity of $O(1)$, i.e., it is independent of the operands' word length $N$ [12]. Conventional adders which are executing binary operands like a carry-look-ahead adder or a conditional-sum adder show a run time complexity in $O(logN)$, or in $O(N)$ like a ripple-carry adder. In order to avoid the generation of carry chains in an addition, a balanced number representation is used in an SD adder. That means that the number of negative digits is the same as the number of positive digits. Since in this

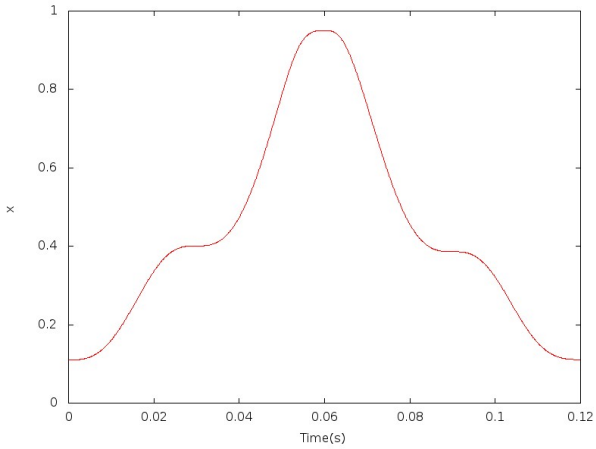


Figure 5. The course of the state variable $x$ of a memristor cell

paper the focus is not on the computer arithmetic side, the functionality of an SD addition is explained by words, by presenting the basic formulae to execute a carry-free addition, and by examples to demonstrate the principle.

First, we investigate roughly the cost of an addition of two SD numbers. The addition is carried out in two steps. In the first step an intermediate sum vector, $z$, and a carry vector, $c$, is calculated. In the second step the two vectors, $z$ and $c$, are post-processed to determine the final sum vector, $s$, which is also given in an SD number representation.

Adding two SD numbers requires a base $r \geq 3$. For the digits, $sd_i$, $0 \leq i < N$, of a $N$ digit long SD number holds $sd_i \, \epsilon \{-(r-1) \ , -(r-2), ..., 0, ..., r-2, r-1\}$. Equation (10) shows the calculations that have to be performed in each digit position $i$ to determine the vectors $z$ and $s$ for two SD input operands $x$ and $y$. The term $c_i \cdot r_i$ in (10) is a so-called correction term. That correction term secures that $z_i \leq |r-2|$. This prevents that the possible addition of a carry bit will cause an additional carry to the left neighboured position. To compensate the addition/subtraction with $c_i \cdot r_i$ it is necessary that a carry bit, either +1 or -1, has to be added in the next left neighboured digit position $i+1$.

$$
\begin{aligned}
z_i &= x_i + y_i - c_i \cdot r \\
c_i &= \begin{cases} +1, & \text{if } (x_i + y_i) > r-2 \\ -1, & \text{if } (x_i + y_i) < -(r-2) \\ 0, & \text{if } -(r-2) \leq (x_i + y_i) \leq r-2 \end{cases} \\
s_i &= z_i + c_{i-1}
\end{aligned}
\tag{10}
$$

Table I shows an example calculation to a base $r = 3$ with a word length $N = 4$. The two SD numbers to add are $1 \cdot 3^3 + (-2) \cdot 3^2 + 1 \cdot 3^1 + 0 \cdot 3^0 = 12$ and $0 \cdot 3^3 + (-1) \cdot 3^2 + 1 \cdot 3^1 + 1 \cdot 3^0 = -5$. The correction term for the second and the first digit positions are $+3$ and $-3$, resp., since $x_2 + y_2 = -3 < -(r-2) = -1$ and $x_1 + y_1 = 2 > (r-2) = 1$. In Table I the carry vector, $c_{i-1}$, is already shifted to left for a better illustration of the final addition in (10). As expected the sum is $s = 0 \cdot 3^3 + 1 \cdot 3^2 + (-1) \cdot 3^1 + 0 \cdot 3^0 = 7$.

An evaluation of this addition scheme to a base $r = 3$ and a realisation with memristors yields the following statements.

 ISBN: 978-1-61208-389-6

TABLE I. ADDING TWO SD NUMBERS $x$ AND $y$ IN TWO STEPS.

| | | | | | | |
|---|---|---|---|---|---|---|
| | 1 | -2 | 1 | 0 | $= 12$ | $x$ |
| | 0 | -1 | 1 | 1 | $= -5$ | $y$ |
| step1 | 1 | 0 | -1 | 1 | | $z$ |
| | -1 | +1 | 0 | . | | $c_{i-1}$ shifted to left |
| step 2 | 0 | +1 | -1 | 1 | = 7 | s |

For the required signed digits, $sd_i$, holds $sd_i \epsilon [-2, 2]$ That means five levels are to store in a single physical memristor cell. That is not impossible in principle but there are two more states to realise than in a ternary computer making the interface circuitry to the memristor cells more difficult to realise. Furthermore a larger effort for the coding is necessary. Three bits are required to code the five possible values, yielding $\binom{8}{5} \cdot 5! = 6720$ different possibilities to assign five 3-bit long code words for $x_i, y_i, z_i$, $0 \leq i \leq 2$, to cover the values $\{-2, \ldots, +2\}$. Table II looks like nearly as the function table from which the Boolean logic is to derive that allows calculating the sum code bits.

TABLE II. Scheme for function table to determine (10).

| $x_2$ | $x_1$ | $x_0$ | $y_2$ | $y_1$ | $y_0$ | 2 bits for $r \cdot c_i$ | $z_2$ | $z_1$ | $z_0$ |
|---|---|---|---|---|---|---|---|---|---|
| 5 entries | | | | | | 0, 1 or -1 | ... | ... | ... |
| for $[-2, +2]$ | | | | | | 0, 1 or -1 | ... | ... | ... |
| ... | | | | | | 0, 1 or -1 | ... | ... | ... |
| 6720 possibilities | | | | | | 0, 1 or -1 | ... | ... | ... |
| for 25 entries | | | | | | ... | ... | ... | ... |

To the best knowledge of the author it is not known if a minimal coding for such a function exists. However, it is to expect that the gate logic is much higher compared to an alternative operation mode, which is preferred in this paper.

This refers to an arithmetic that allows to operate a SD number, $SD$, with a binary number, $B$. For such an operation the coding for the SD number producing a minimum gate effort is known. Already in the 1990s Muller and Duprat [13] published the efficient coding scheme shown in Table III for the positive and the negative channel of a SD number, $SD$. This allows ternary coding, what in addition simplifies the realisation of the interface to store the SD value in a memristor cell since only three states are to distinguish. Furthermore the necessary gate logic, shown in (11), to calculate the intermediate sum bits, $z_i$, the carry bits, $c_i$, and the positive and negative channel of the sum, $s$, is comparatively simple. An example calculation for the addition of a SD number, $a$, and a binary number, $B$, presents Table IV. The shown bits for $z_i$, $c_i$, $SD_i^+$, and $SD_i^-$ are calculated by applying all the equations given in (11) in each digit position. The technique to avoid the carry propagation in this case is to generate in the intermediate sum vector only 0's and -1's, and in the carry vector only 0's and +1's. This excludes that in the second step two positive or negative 1's will meet in one digit position to produce a further carry.

TABLE III. Digit coding of a binary SD number.

| $SD^+$ | $SD^-$ | Value for $SD$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $\bar{1}$ |
| 1 | 0 | 1 |
| 1 | 1 | not defined |

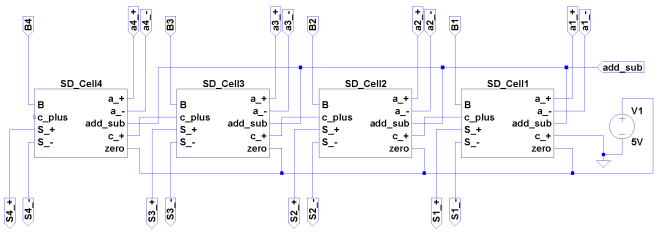


Figure 6. Schematic for a 4-digit SD adder that can add/subtract a SD number $a$ and a binary number $B$

$$
\begin{aligned}
c_i^+ &= SD_i^+ \vee \left(B_i \wedge \overline{SD_i^-}\right) \\
z_i^- &= \left(SD_i^+ \vee SD_i^-\right) \oplus B_i \\
s_i^+ &= \overline{z_i^-} \wedge c_{i-1}^+ \\
s_i^- &= \overline{c_{i-1}^+} \wedge z_i^-
\end{aligned} \tag{11}
$$

A detailed analysis how the Boolean equations from (11) can be derived and how they are mapped onto an SD adder using memristors as in-/outputs can be found in [13][8]. There is also shown a detailed verification of the functioning of the SD adder by SPICE simulations [8]. A schematic of that SD adder is shown in Figure 6. In each module denoted as SD cell the equations shown in (11) are carried out on the SD number $a$, and the binary number B for word length of $N = 4$. The signal *addsub* determines if an addition or a subtraction is carried out. A subtraction can be easily reduced to an addition according to (12). The inverse of an SD number can be simply determined by exchanging the negative with the positive channel. The signal *zero* clears the binary input, if it is set to logical 0, i.e., an addition is carried out with zero. That is necessary in case of a multiplication if the multiplier bit is zero. If $zero$ is set to 1, the corresponding bit of the second operand $B$ is passed. The output sum, $s$, as well as the input, $a$, are SD numbers, which have a positive and a negative part and can be stored in memristor cells. Details how this adder is operating can be found in [8]. There is also included a comparison that proves the performance benefits of an SD adder compared to existing methods like ripple-carry or carry-look-ahead adders. In this paper, the focus is laid on how combinations of such SD adders can be used for a much easier operation of two SD numbers, which is reduced to subsequent series of additions of a SD number and a binary number. The price for that reduction is only a small higher latency compared to the direct addition of two SD numbers shown in Table I.

TABLE IV. Example of a carry-free addition for two positive integers, A and B, using binary coded SD numbers.

| | | | | | | |
|---|---|---|---|---|---|---|
| a | . | 0 | $\bar{1}$ | 0 | 1 | $= (-3)_{10}$ |
| B | . | 1 | 0 | 0 | 1 | $= (9)_{10}$ |
| z | 0 | $\bar{1}$ | $\bar{1}$ | 0 | 0 | |
| c | 1 | 0 | 0 | 1 | . | |
| s | 1 | $\bar{1}$ | $\bar{1}$ | 1 | 0 | $= (16 - 8 - 4 + 2)_{10} = (6)_{10}$ |

$$SD - B = -(-SD + B) \tag{12}$$

 ISBN: 978-1-61208-389-6

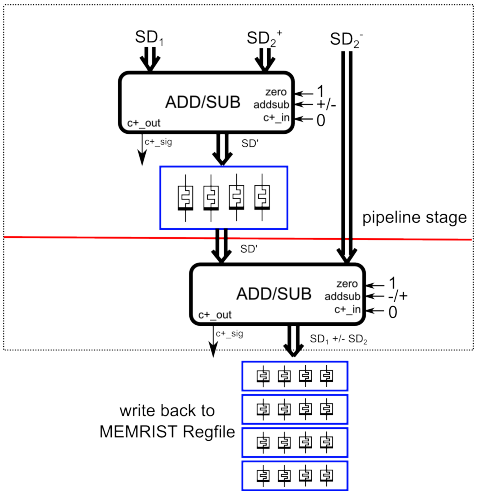


Figure 7. Schematic for adding/subtracting two SD numbers using memristors as pipeline register

## IV. MEMRISTOR BASED PIPELINE STRUCTURES

An addition between two SD numbers, $SD_1$ and $SD_2$, can be decomposed in two steps, in which in the first step $SD_1$ is processed with the positive channel of $SD_2^+$ and in the second step the negative channel is subtracted from the result of the addition (13). In the case of a subtraction the signs of the operators signs have to be inverted (+ becomes - and vice versa).

$$SD_1 \pm SD_2 = ((SD_1 \pm SD_2^+) \mp SD_2^-) \tag{13}$$

Figure 7 shows the schematic for a corresponding arithmetic circuit. Two of the addsub modules shown in Figure 6 are necessary with an appropriate interface circuitry, not shown in Figure 7, to store the first result in a memristor register. This allows to pipeline the calculation before the second addsub module comes to execution.

Figure 8 shows a scheme for a multiplication of a SD number $a$ with a binary number $B$. A multiplication of operands with $N$ digits is reduced back to $N$ subsequent additions/subtractions. Each addsub module forms a pipeline with memristors to store the partial product digits. In the first three steps the product digits, $P[0]$ to $P[3]$, are produced before in the last pipeline step the final product digits $P[4]$ to $P[7]$ appear at the output of the last addsub module and all digits can be stored in a memristor register file. The digits of the SD operand $a[i]$ determine in each pipeline stage, $i$, if either an addition ($a[i]- = 0$) or a subtraction ($a[i]- = 1$) has to be carried out with the partial product and the binary number $B$, or if in case $a[i] = 0$ no addition takes place.

Multiplication of two SD numbers, $SD_1 \cdot SD_2$ can be traced back to a hybrid operation scheme that executes a SD number and a binary number (14) as it was done for the addition/subtraction, see Figure 9. First, two intermediate products, $Prod_1$ and $Prod_2$, are generated with two concurrently operating circuits from Figure 8. Afterwards these two products, given in SD number representation, are subtracted with a two step circuit analogue to the addition/subtraction shown in Figure 7. If no multiplication has to be carried out

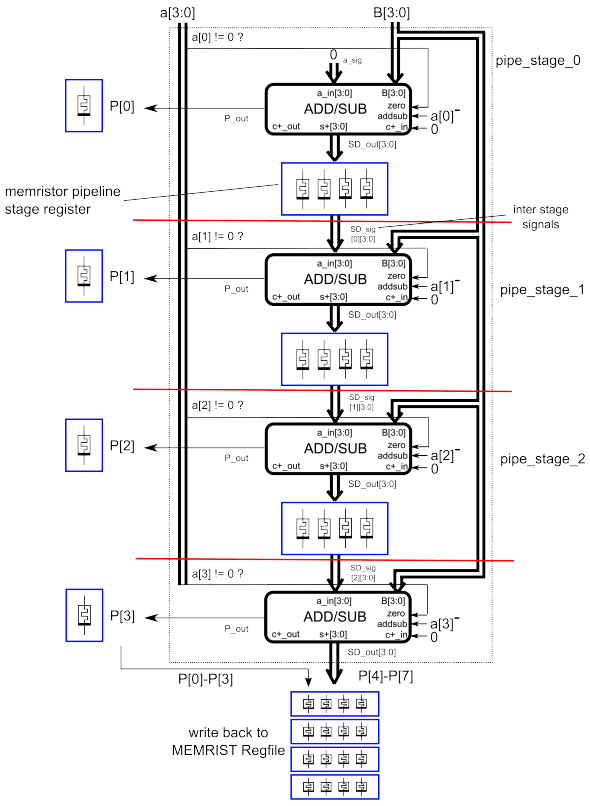


Figure 8. Memristor-based pipelining for the multiplication of a SD number, $a$, and a binary number $B$

the two multiplier pipelines can be used for two concurrent additions/subtractions. Therefore, Figure 9 represents the core of a future memristor multi-threading architecture, which the author intends to realise with largely homogeneous pipeline structures.

$$\begin{aligned} SD_1 \cdot SD_2 &= (SD_1 \cdot SD_2^+) - (SD_1 \cdot SD_2^-) \\ &= Prod_1 - Prod_2 \end{aligned} \tag{14}$$

## V. EVALUATION OF MEMRISTOR-BASED PIPELINE STRUCTURES

The functionality of all proposed pipeline architectures was verified by simulations in SystemC. In this section the pipeline architectures are compared with regard to latency, expressed in gate delays, achievable throughput, and necessary hardware effort, which is measured in numbers of necessary transistors and memristors (Table V).

First, the add/sub pipeline from Figure 7 is analysed. If $\Delta$ corresponds to one gate delay, according to (11) an add-sub module requires $4\Delta$ plus the time, $t_{conv}$, for converting the digital signals by a number of voltage oscillations (see Figs. 3 and 5) to store the SD adder outputs in multi-bit memristor cells at the end of the first pipeline stage. In [8], $N \cdot 62$ transistors were determined for the used SD adder, shown in Figure 6. Additionally, the number of transistors for the conversion, $trans_{conv}$, is to add for each of the $N$ memristors. An adder that adds directly two SD numbers using

TABLE V. Evaluation of the memristor-based pipeline architectures. Time to write back to MEMRIST regfile is not considered

| | latency | # memristors | # transistors |
|---|---|---|---|
| Addition/Subtraction | $4\Delta + t_{conv}$ | $N$ | $N \cdot (124 + trans_{conv})$ |
| SD-binary multiplier | $N \cdot (4\Delta + t_{conv})$ | $N^2$ | $N^2 \cdot (62 + trans_{conv})$ |
| SD-SD multiplier | $(N+2) \cdot (4\Delta + t_{conv})$ | $2N^2 + 4N$ | $(2N^2 + 4N) \cdot (62 + trans_{conv})$ |

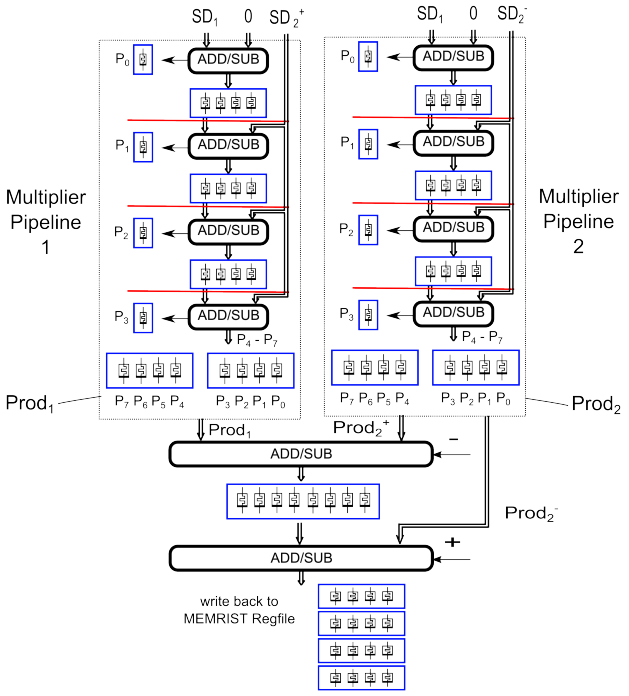


Figure 9. Memristor-based pipelining for the multiplication of two SD numbers with a digit length of four.

a base $r = 3$ would show a latency of only $2\Delta$. However, it is to expect that this requires much more transistors for logic and conversion, since five and not three multiple memristor states have to be stored. Therefore, the conclusion is drawn that the double latency time of a SD-binary adder is acceptable.

Counting the number of memristors and addsub modules of the SD-binary multiplier in Figure 8 for the case $N = 4$ and deducing from that the generic case yields the numbers shown in Table V for latency and hardware effort. The SD-SD multiplier consists of two multipliers and two further SD adders with $2N$ digit length leading to the values entered in the last row of Table V. The throughput for all architectures is equal to the reciprocal of the latency in one stage, $1/(4\Delta + t_{conv})$. Access times for memristors in the range of $0.3\,ns$ make realistic to achieve a clock frequency of more than $1\,GHz$.

## VI. Conclusion

Memristors and their capabilities to store multi-bits in a single cell allow the set-up of fast CMOS compatible carry-free arithmetic circuits. The presented analysis shows that the effort for the realisation of ALUs with the proposed hybrid SD-binary multiplier pipeline approach and pipeline memristors is quite moderate and to prefer versus a pure SD-SD adder arithmetic. The half latency time of the hybrid solution is offset against the pure SD approach that would cost much more hardware. Future work requires the design of a corresponding control unit to feed the pipeline with a stream of instructions and a detailed analysis how to realise efficiently an interface circuit to the memristors.

## Acknowledgment

The author thanks Jonathan Martschinke who wrote the numerical simulation program of memristor's behaviour that was used to generate Figs. 3 to 5.

## References

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

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

[3] B. Linares-Barranco and T. Serrano-Gotarredona, Memristance can explain spike-time-dependent plasticity in neural synapses. Nature Precedings, March, 2009, hdl:10101/npre.2009.3010.1.

[4] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, Memristor-based IMPLY logic design procedure. Proc. 29th IEEE International Conference on Computer Design (ICCD), October, 2011, pp.142-147.

[5] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, Memristor-based MRL Memristor ratioed logic. Proc. 13th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA), August, 2012, pp. 1-6.

[6] I. Vourkas and G. Ch. Sirakoulis, Memristor-based combinational circuits: A design methodology for encoders/decoders. Microelectronics Journal , vol. 45, no. 1, January, 2014, pp. 59–70.

[7] L. Zheng, S. Shin, and S.-M. S. Kang, Memristor-based ternary content addressable memory (mTCAM) for data-intensive computing. Semiconductor Science and Technology, vol. 29, no. 10, 2014, 104010 (10pp).

[8] 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, 2014, 104008 (13pp).

[9] Y. N. Joglekar and S. J. Wolf, The elusive memristor: properties of basic electrical circuits. Eur. J. Phys., vol.30, no. 661, January, 2009, pp. 1-24, arXiv:0807.3994 v2 [cond-mat.meshall].

[10] Z. Biolek, D. Biolek, and V. Biolkov, SPICE model of memristor with nonlinear dopant drift. Radioengineering, vol. 18,no. 2, June, 2009, pp. 210-214.

[11] N. R. McDonald, R. E. Pino, P. J. Rozwood, and B. T. Wysocki, Analysis of dynamic linear and non-linear memristor device models for emerging neuromorphic computing hardware design. IEEE IJCNN, 2010, pp. 1-5.

[12] A. F. González and P. Mazumder, Redundant arithmetic, algorithms and implementations. INTEGRATION, the VLSI journal, vol. 30, no. 1, 2000, pp. 13-53.

[13] J. Duprat and J.-M. Muller. Fast VLSI implementation of CORDIC using redundancy. in E.F. Depreterre, and A.-J. van der Veen (eds.): Algorithms and Parallel VLSI architectures, Elsevier, vol. B, June, 1991, pp. 155-164.

 ISBN: 978-1-61208-389-6