A memristive circular buffer for real-time signal processing

Christos Sichonidis*, Ioannis Vourkas†, Nikolaos Mitianoudis* and Georgios Ch. Sirakoulis*

*Department of Electrical and Computer Engineering
Democritus University of Thrace, Xanthi, Greece
Emails: {csichoni, nmitiano, gsirak}@ee.duth.gr

† Department of Electrical Engineering
Pontificia Universidad Católica de Chile, Santiago, Chile
Email: iovourkas@uc.cl

***Abstract*—Thanks to their ability to store information in a continuous (analog) form, memristors are termed as well-suited for several real-time signal processing tasks. In this context, here we present a memristive circular buffer, using memristor and its multi-bit storage ability to temporarily store encoded information in a compact form, thus improving the area performance as well as the delay and energy consumption of the circuit, compared to conventional designs. We introduce the arithmetic encoding principles for the proposed circuit, explain the en/decoding mechanisms for the memristors-based data-management operations, and finally present the target application. For our simulation-based study we used a threshold-type device model of a bipolar voltage-controlled memristor.**

I. Introduction

Arithmetic encoding is a form of entropy encoding used in lossless data compression. It is defined as a variable-length message coding based on the frequency of every character. The message is represented by a fraction which is the repeated offset-plus-product reduction of the range (offset) and probability (product) of each character [1]. The offset can be practically represented by variable physical sizes which may take any value between two outliers. The characters, or symbols, as we may refer to them that the message consists of, are both encoded/decoded serially by dividing the available range in smaller intervals and then picking the one that represents the encoded/decoded symbol, which then becomes the range for the next symbol. The coding process is stopped when the offset becomes smaller than the noise level of the system, making it difficult (or impossible) to distinguish the different symbols. The encoding is based on the a priori knowledge of the incidence of the symbols in the message, so that the maximum number of symbols is encoded.

In our case, we correspond the variable physical size to the variable resistance of a memristor, taking values within the range $[R_{ON}, R_{OFF}]$ [2]. We particularly apply a constant input current to the memristors, so we have a resistance proportional to the corresponding voltage drop on its terminals, which is easy to measure (and thus manage the size) and feed to the rest of the circuit components. Via the employed arithmetic encoding operations, the message is encoded and stored in memristors, arranged in a circuit known as *circular buffer* (*CB*).

The term CB is frequently used in software programming and it refers to an area of memory used to store a continuous stream of data, starting again at the beginning of the buffer after passing its end. Usually separate processes access the CB for reading and writing and separate read- and write-pointers are maintained, which are not allowed to pass each other's position. However, implementing CBs in hardware (HW) could be advantageous in terms of computing speed, improving significantly the execution of repetitive digital signal processing (DSP) tasks in complex DSP algorithms.

To this end, in this paper we present a novel implementation of a CB circuit using memristor technology in multi-bit memory cells for arithmetic coding. Thanks to multi-bit storage, i.e. fewer necessary memory cells, the memory addressing circuitry results simpler, the read/write operations become faster, and the circuit area is significantly reduced. We particularly build upon the work presented in [3] and invoke the proposed here circuits in a novel CB circuit design. In our implementation we rely on threshold-type switching bipolar memristors [2], since this type suits best our purposes. In fact, such memristors are unaffected by noise level lower than their switching voltage threshold, below which the induced memristance change can be considered negligible, hence guaranteeing system stability and preventing from bit losses via disturbed resistive states.

II. Threshold-based Memristor

In his seminal 1971 paper [4], Leon Chua first spoke about the existence of the fourth fundamental passive circuit element (besides the resistor, capacitor and inductor) which he called a memristor (short for memory-resistor). The memristor (here used to describe both an ideal memristor as well as a generalized memristive device [5]) is a passive two-terminal electronic device whose behavior is described by a nonlinear constitutive relation: $v = R(x,i) \cdot i$ between the voltage drop at its terminals v and the current flowing through the device i, where: $dx/dt = f(x,i)$ and x denotes a set of internal state-variables, whereas the nonlinear function $R(x,i)$ is called memristance and has the unit of Ohms (Ω). When the applied voltage is turned off, a memristor still remembers how much voltage was applied before and for how long; thus presenting memory of its past, a property which qualifies it as a fundamental circuit element.

Threshold-type switching is closer to the actual behavior of most experimentally realizable memristive devices. Throughout this work, in all simulations we use a threshold-type model of a voltage-controlled bipolar memristor [2], which attributes memristance-switching to quantum tunneling. The model is based on the assumption that the switching-rate is small below (fast above) a voltage threshold (namely V_{SET} or V_{RESET}). The model permits asymmetric thresholds ($|V_{SET}| \neq |V_{RESET}|$) and different tunneling distance change-rates for SET and RESET operations. The values of the parameters of the model were set as $\{a, b, c, m, f_o, L_o\}$ = $\{10, 500, 1, 000, 0.1, 82, 310, 5\}$ with $\{R_{ON}, R_{OFF}\} = \{2, 200\}k\Omega$ and $\{V_{SET}, V_{RESET}\} = \{1.5, -1.5\}$V. State-reading of memristors relies on using a voltage lower than the voltage thresholds, hence it does not affect (or affects negligibly) the stored resistive state. For our particular purposes, based on the *state-dependent Ohm's Law* in (1), in this work the selected voltage-controlled memristor model was converted to its current-controlled version, i.e. considering current as the input signal instead of voltage, without loss of generality.

III. Fundamentals of Arithmetic Encoding

This section presents the processes used in arithmetic coding on memristive devices. Such processes are the basis for the proposed CB circuit, transferring data to/from memory cells.

A. Write operation

In our work, the write operation to a memristor device consists of three consecutive steps and the corresponding circuit is shown in Fig. 1 [3]. The first step is the creation of a reference voltage on the "Memristor switchbox". Since this is the objective of this operation, when it is accomplished, then the input current flow stops, as shown in the respective simulation results in Fig. 2. For this reason, the reference voltage is driven to the inputs of two comparators. Afterward, the current resistive state of the memristors is read and then a decision is made whether it should be increased or decreased, via the application of a negative (BL') or a positive (BL) current, respectively. To this end, the "write direction" comparator selects the corresponding bit-line. The final step consists in driving the memristors with a constant current until the voltage drop on their terminals becomes equal to the reference voltage, and the "write termination comparator" gives 0V output, i.e. "End" flag signal. The simulation results in Fig. 2 give an insight to what happens to a memristor based on the model of [2] during the write procedure.

According to [3], the criterion for the cease of the encoding procedure is whether the minimum distinguishable voltage V_{min} is smaller than the minimum allowed value, i.e.:

$$V_{min} \geq 2(V_n + V_{drift} + V_{SH}) \tag{1}$$

where V_n is the voltage corresponding to the circuit noise, V_{drift} is the maximum voltage caused by resistive drift due to delayed ending of the write operation, and V_{SH} is the voltage

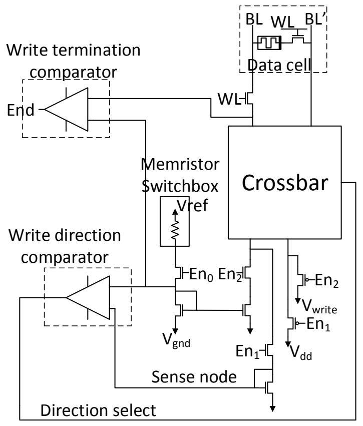


Fig. 1. The write operation circuit (adapted from [3]).

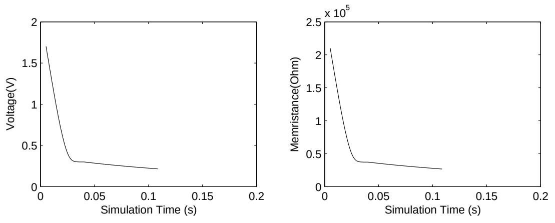


Fig. 2. Simulation results concerning the change of the voltage and resistance of a memristor in time.

error introduced to the system by the sample & hold (SH) component in the read circuit described below and shown in Fig. 3.

B. Read operation

Reading works much like the iterative procedure described in the arithmetic coding introduction. The corresponding circuit is shown in Fig. 3. A constant current of less than 10uA is provided to the target memory cell, which creates a voltage drop analogous to its memristance. The initial maximum (V_{top}) and minimum (V_{bottom}) values, respectively, are given as: $V_{top} = I_{init} \cdot R_{max}$ and $V_{bottom} = I_{init} \cdot R_{min}$. For the n-th symbol, $n \in N$, the read cell voltage V_{read} is compared to the switchbox output V_{rn} which is the borderline between the two symbols and is equal to:

$$V_{rn} = (V_{top} - V_{bottom}) \cdot p_0 + V_{bottom} \tag{2}$$

where p_0 is the probability of the symbol A (the range below V_{rn}), while the probability of the symbol B is $p_1 = 1 - p_0$. Depending on the outcome of this comparison we have: a) when $V_{rn} > V_{read}$ the symbol $S_n = A$ is decoded and $V_{top} = V_{rn}$, b) when $V_{rn} \leq V_{read}$ the symbol $S_n = B$ is decoded and $V_{bottom} = V_{rn}$. Afterwards, V_{rn} is updated according to by (2) and the next symbol S_{n+1} is decoded. The procedure is repeated until all symbols are decoded and then sent to a shift register. The corresponding voltage plot for a four-symbol decoding example is shown in Fig. 4.

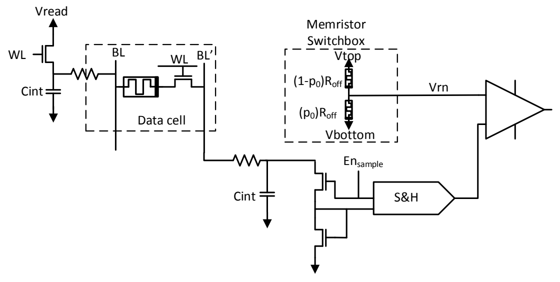


Fig. 3. Read operation circuit (adapted from [3]).

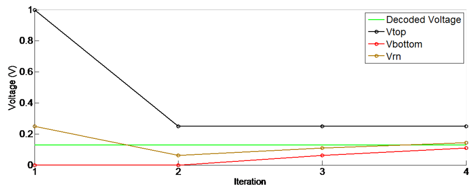


Fig. 4. An insight of the arithmetic decoding. The cell outputs a voltage that represents 4 symbols in 4 iterations, with each one having an upper limit (V_{top}), a lower limit (V_{bottom}) and a borderline (V_{rn}). Their comparison via the Decoded Voltage defines each time the symbol. The message decoded is “ABBA”.

IV. The Circular Buffer

In this section we describe the proposed implementation of a memristive CB. In this context, all the circuits presented previously in section 2 are now used. In the DSP scientific field, it is needed to create a delayed version of the input signal which interferes with it either destructively or constructively. Nevertheless, it is quite common to have HW restrictions on the memory size and the data access time. A circular buffer manages to use effectively every cell in its dedicated memory. Its total size is equal to the maximum delay.

The buffer consists of $N = 2^x$, with $x \in \aleph$ cells, so that we can use a binary address decoder. Every storage cell in the buffer stores data in the memristor. While buffer is functioning, cells are serially read and written as follows:

1) For a new incoming stream, the access transistor (see Fig. 5) of the cell on the “write position” is switched on via the WL, as shown in Figs. 1 and 3, using a $log2(N) \times N$ decoder for a duration equal to the write time.
2) Then WL is switched off and the decoder switches on the cell transistor on the read position for the read process. The delayed sample is sent to the buffer output.
3) The counter is increased by 1 and waits for the next sample to point the decoder to the next cells

$$\text{Write\_pointer} = \text{mod(iteration + unprocessed\_cells, buffer\_size) +1} \quad (3)$$

$$\text{Read\_pointer} = \text{mod(iteration-1, N)+1} \quad (4)$$

This is a memory structure preferable to e.g. FIFO type, because the latter requires each cell to pass its contents to the next one in every iteration. A block diagram for the components of a memristive CB is given in Fig. 5.

The CB works in a repetitive process which in short is as follows: When the stream data arrive to the write circuitry, the counter provides the decoder with the numbers of the next cells to read/write. The decoder activates the corresponding transistors preventing from interference between signals of the read/write circuitry. The write circuitry encodes the input data. Then the written cell is deselected and the read circuit goes on to collect the data from their cell, accessed via the decoder with a delay of N elements:

$$D = T_D f_s \quad (5)$$

where T_D is the delay in seconds and f_s is the sampling frequency. Collected data are sent by the read circuit to the rest of the system. Among the advantages of using memristors as memory cells, are the read and write speed, the non-volatility, and also their ability to work both in digital and analog manner. Thanks to their inherent analog nature, their memristance may take any intermediate value within the $[R_{ON}, R_{OFF}]$ range, limited only by the accuracy of signal encoding/decoding and propagation of the system noise which is added from them. This provides us with the option of avoiding the procedure of quantization and therefore the disturbance that causes the mapping of signal on a standard number of different levels [6]. In this case, the write procedure remains exactly as it was presented before since it is implemented in analog circuit. On the other hand, the read procedure does not keep the intermediate values to find the decoded symbols, but only the final value of the voltage, which should be proportional to the device memristance.

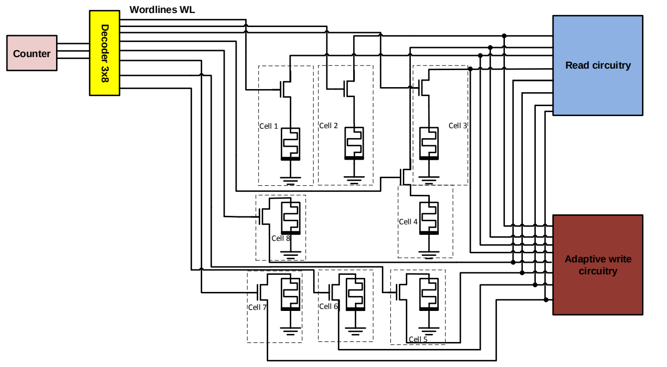
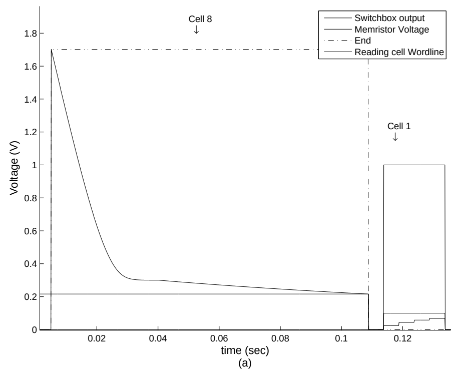


Fig. 5. Block diagram describing the main parts of an 8-element memristive circular buffer. The memory cells are numbered in the order in which they are read during simulation.

V. Simulation Results

The simulation-based validation of the proposed CB of Fig. 5 was done using Matlab, for a conservative 4-character encoding scheme and multi-bit storage in memristors. The alphabet was the simplest possible compiled by two symbols A, B with probabilities $p_0 = 0.25$ and $p_1 = 0.75$, respectively. The delay elements in (9) have D=8 so that the data have the maximum possible delay and the read memory cell in a particular moment is right afterwards overwritten. In the simulation results shown in Fig. 6, blue line symbolizes the voltage on all memristors for both reading and writing and, in order to distinguish them, the End signal from Fig. 1 and the



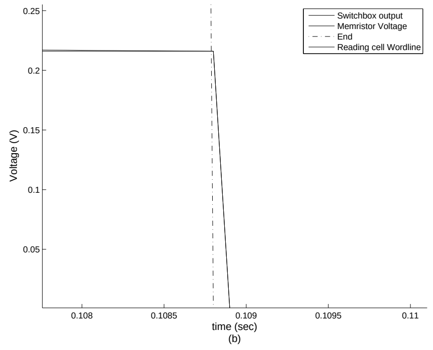


Fig. 6. (a) The diagram of the first iteration of the circular buffer function. Cell 8 is written and cell 1 is read. (b) The write operation stops with a short latency after the End signal is pulled down. The caused deviation from the target value is called voltage drift.

Wordline signal from Fig. 3 are also given. During simulation the words encoded take progressively values closer to V_{top}. Both write and read circuits are simulated to implement the operations described previously and to maintain all their characteristics (see Fig. 6b).

As we see in Fig. 7, the eight last encodings are conducted through negative voltage, to augment the cell resistance to the word level. Reading operations require smaller voltages (maximum 100mV) applied for $\Delta t = 5$ms, while writing highly depends on the initial value and the switchbox output. When one of them is between the voltage thresholds, writing can last up to $\Delta t =$100ms (e.g. the two encodings of Cell 8), or be faster even than reading (e.g. for Cell 7). It is worth noting that, due to the threshold-type nature of memristors, the impact of the read voltage on the device is negligible.

Finally, the simulation characteristics are presented along with the results of decoding in each iteration. Disparity regards the probabilities of the encoded symbols and V_{min} given by (1): Disparity = 0.5 and $V_{min} = 34.8375$ mV. For the first 7 iterations the cells have not been edited and the memristor resistance stays in the R_{OFF} condition and then the written data begin to appear, due 8-element delay. The repetitive application of the aforementioned procedure creates a data flow that is read out of the buffer and back to the DSP system, identical to its input with only difference the 28-symbol delay (7 cells delay by 4 encoded symbols each), which is the desired result and in a more general context, the function of a buffer. The decoder ensures that the buffer will overwrite its old data when full, without any extravagant pointer circuits making it not only circular, but also cost-effective.

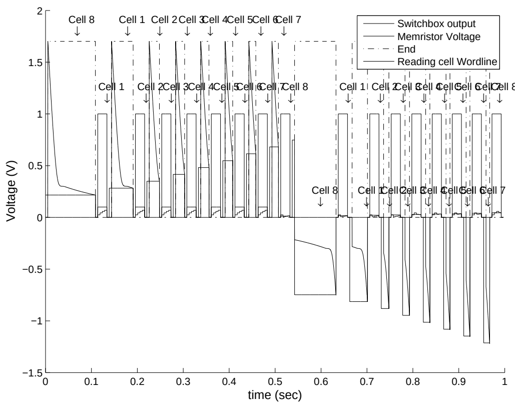


Fig. 7. The timeline of a 16-iteration simulation. Arrows indicate the cell access during every procedure. When End signal is up indicates a write process, while Wordline is used for reading.

TABLE I
READING OPERATION RESULTS IN FORM: #ITERATION[RESULT].

#1[BBBB]	#2[BBBB]	#3[BBBB]	#4[BBBB]
#5[BBBB]	#6[BBBB]	#7[BBBB]	#8[ABBA]
#9[ABBB]	#10[ABBB]	#11[ABBB]	#12[BAAB]
#13[BABA]	#14[BABB]	#15[BABB]	#16[BBAA]

VI. CONCLUSIONS

A novel memristive circular buffer architecture is designed and simulated, along with the peripheral read/write/access circuits. Circuit functionality is validated through simulations based on a threshold-type memristor model, leading to more stable and noise-tolerant system. Simulation results prove functionality and proper cooperation among the different circuit parts. Future work concerns using the proposed system to execute DSP tasks in complex DSP algorithms in HW.

ACKNOWLEDGEMENT

The authors acknowledge support to and wish to thank European ICT COST Network MemoCIS: IC1401

REFERENCES

[1] P.E. Black, *Dictionary of Algorithms and Data Structures*, 1998.

[2] I. Vourkas and G.Ch. Sirakoulis, *A Novel Design and Modeling Paradigm for Memristor-Based Crossbar Circuits*, IEEE Trans.on Nanotechnology, vol. 11, pp.1151-1159, 2012.

[3] R. Patel and E.G. Friedman, *Arithmetic Encoding for Memristive Multi-Bit Storage*, 2012.

[4] L. Chua, *Memristor-The Missing Circuit Element*, IEEE Trans. on circuit theory, Vol. CT-18, No. 5, pp.1151-1159, 1971.

[5] D.B. Strukov, G.S. Snider, D.R. Stewart and R. Stanley Williams, *The missing memristor found*, Nature, Vol. 453, pp. 80-83, 2008.

[6] B. Widrow and I. Kollar, *Quantization noise in Digital Computation, Signal Processing Control and Communications*, Cambridge University Press, Cambridge, UK, 2008.