Article

Stochastic Computing Emulation of Memristor Cellular Nonlinear Networks

Oscar Camps [1,‡] 0000-0003-0068-3422, **Mohamed Moner Al Chawa** [2,‡] 0000-0001-8886-4708, **Stavros G. Stavrinides** [3,‡] 0000-0001-8484-1402 **and Rodrigo Picos** [1,4,*] 0000-0002-9167-6422

[1] Industrial Engineering and Construction Department, Balearic Islands University, Palma Mallorca, Spain; oscar.camps@uib.es, rodrigo.picos@uib.es
[2] Technical University of Dresden, Dresden, Germany; mohamad_moner.al_chawa@tu-dresden.de
[3] School of Science and Technology, International Hellenic University, Thessaloniki, Greece; s.stavrinides@ihu.edu.gr
[4] Health Institute of the Balearic Islands (IDISBA), Palma Mallorca, Spain
[*] Correspondence: rodrigo.picos@uib.es
[‡] These authors contributed equally to this work.

Abstract: Cellular Nonlinear Networks (CNN) are a concept introduced in 1988 by Leon Chua and Lin Yang as a bio-inspired architecture, capable of massively parallel computation. Later on, CNN have been enhanced by incorporating designs that incorporate memristors to profit from their processing and memory capabilities. In addition, Stochastic Computing (SC) can be used to optimize the quantity of required processing elements; thus it provides a lightweight approximate computing framework, quite accurate and effective, though. In this work, we propose utilization of SC in designing and implementing a memristor-based CNN. As a proof of the proposed concept, an example of application is presented. This application combines Matlab and a FPGA in order to create the CNN. The implemented CNN has then been used to perform three different real-time applications on a 512x512 gray-scale and a 768x512 color image: storage of the image, edge detection, and image sharpening. It has to be pointed out that the same CNN has been used for the three different tasks, with the sole change of some programmable parameters. Results show an excellent capability with significant accompanying advantages, like the low number of needed elements further allowing for a low cost FPGA-based system implementation, something confirming the system's ability for real time operation.

Keywords: Cellular Nonlinear Networks; Stochastic Logic; real time processing; image processing; memristors.

1. Introduction

Cellular Nonlinear Networks (CNN) were introduced by Chua and Yang [1] in 1988, and can be described as a mixture between Cellular Networks and Artificial Neural Networks that can implement a parallel processing universal computer machine. This bio-inspired architecture is able to process in parallel massive amounts of data, thus being specially suitable for image processing, with single ASIC CMOS prototypes already implemented being able to deal with rates up to $3 \cdot 10^4$ frames per second [2].

Memristors have been proposed as a device that may help to implement this kind of circuits ([3,4]), but experimental implementations are still lacking. These devices, memristors, are passive, two-pole elements also introduced by Chua in 1971 [5], as a theoretically possible basic circuit element. In 2008, Strukov et al. [6] realized their ReRAM devices were, actually, a kind of memristor. There have been many groups dedicated to create either devices or emulators, ever since. One of the more classical mathematical memristor description, including memconductance G can be written as:

$$i(t) = G(Q) \cdot v(t) \tag{1}$$

where Q (also known as *charge*) is the integral over time of the current i:

$$Q(t) = \int^t i(t)dt \qquad (2)$$

Notice that the requirement for the device to be a memristor is mapped to the requirement for the characteristics of the device to be dependent on some internal variables, as will be further discussed below, plus some fingerprints [7,8].

One of the main problems for using memristors into circuits is that they are not yet readily available for implementation in usual technologies. In this paper we use Stochastic Computing to implement a fully digital realization of a CNN using memristors. To do so, we have used a memristor emulator presented in [9], as well as a Stochastic Computing implementation of a CNN using it, as in our previous work [10], where a simpler implementation was presented operating exclusively on gray images. The notion of Stochastic Computing (SC) was introduced by Von Neumann during the 50s [11] as a theoretical framework to explain how (relativelly) accurate results could be obtained using imprecise systems. In this framework, time and accuracy are balanced in a trade off. It was later popularized by Gaines [12], and it has found a niche in approximate computing. There are many examples in the literature for diverse applications, such as data compression [13], chaotic equation calculation [14], data mining [15], FFT computation [16], control [17], image processing algorithms [18,19], or also A/D conversion [20]. As of today, it seems probable that it may conquer an important share in edge computing applications, since it can decrease energy consumption and circuit complexity and area overhead for low numbers of bits. However, this comes to a price, since the time required to perform the operation also increases exponentially with the number of bits. This, in turn, may increase the total energy consumption when the number of bits exceeds 16-17 [21,22]. There are, however, techniques that allow this problem to be alleviated, and make this competitive even for higher bit-numbers [22].

Within this paradigm, strings of 2^N binary digits $b_i \in [0,1]$ are used to represent real numbers p. These strings of digits are called stochastic computing numbers (SCN) and their mean value corresponds to the number p represented [23]. Thus, the absolute value ranges for any implementation of SCN must fall within [0,1]. There are two distinct possibilities to map a real number to SCN: the first one is from the domain [0..1], and the second one corresponds to [-1..1].

Once the selected mapping has been decided, the different SCN operations can be easily implemented, requiring only the use of fundamental logic gates or very simple digital circuits. As an example, in the case of the [0..1] domain, multiplication of SCNs are implemented using an AND gate; in the second case, the [-1..1] domain, this same operation demands using a XNOR gate.

The full system we have implemented is depicted in Fig. 1. The first part is processed in the computer, where the images are read using a Matlab script and converted to gray scale if needed. The resulting image is then sent to the FPGA board, which is used as an accelerator and connected using the FPGA-in-the-loop methodology that allows to integrate it in a seamless way. The FPGA does the processing using a massively parallel implementation of the proposed Stochastic Memristive Cellular Nonlinear Network, and the result is then read back into the computer and represented to the user, along with the entropy of the image and the rms error if needed.

The paper is structured as follows: after this introduction, the basics of memristors and Cellular Nonlinear Networks are presented in section 2, which is being used in Section 3 to implement the basic CNN cell in Stochastic Computing. Section 4 presents the results obtained using three different pictures (two gray, on color) with three different sets of parameters. These three sets of parameters allow the CNN to perform three different operations on the images: storing, edge detection, and image improvement. Finally, Section IV concludes the paper.
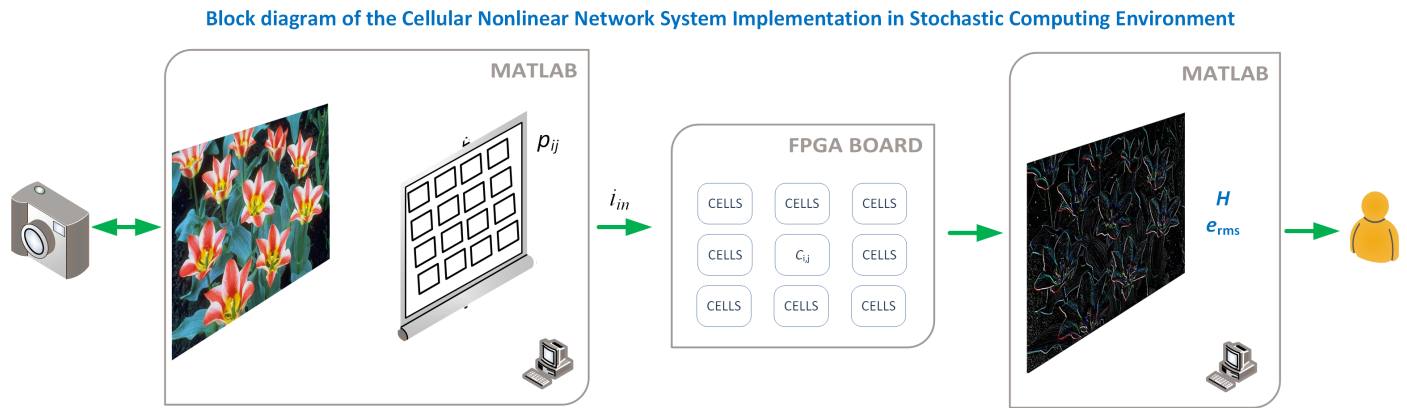
Figure 1. Conceptual depiction of the system, showing the tasks assigned to Matlab and those performed by the FPGA. The FPGA and Matlab are used jointly by using the FPGA-in-the-loop tool from Matlab, where the VHDL code is automatically generated, uploaded, and integrated with the main script at the computer.

2. Memristive Cellular Nonlinear Networks

2.1. Memristors and memristive systems modeling

Among the possible theoretical descriptions of memristive systems, Corinto *et al*. present in [24] a very complete framework to study memristors and, in general, systems that may present memristive behavior. They propose using both the classical description using voltage and current, but they also discuss the flux–charge (φ-q) approach.

The memristive systems can be classified according to how far they are from ideality. Following the taxonomy proposed in [7], there are three distinct possibilities: the ideal, the generic, and the extended memristor. This extended categorization was a theory requirement, needed to cover the description of pinched, hysteretic behaviours found in numerous new various elements.

The Most general class of memristors are the extended memristors. The memristors belonging to this class are described by extra internal state variables (in addition either to current and voltage, or to φ and q). As an example, Eqs. (3) to (5) implement the case of flux-controlled memristors:

$$i = G(\varphi, v, \mathbf{x}) \cdot v \tag{3}$$

$$\dot{\mathbf{x}} = \mathbf{g}'(\varphi, v, \mathbf{x}) \tag{4}$$

$$\dot{\varphi} = v \tag{5}$$

This way, an extended memristors has a memristance M represented by the nonlinear memconductance G (or, more accurately, its inverse) in Eq. (3), where φ is the flux, and v is the voltage between the terminals of the memristive device. The extra variables are grouped into the vector $\mathbf{x}$, and they may comprise different physical magnitudes depending on the specific memristive system; as examples, we can mention the radius of a conducting filament, the internal temperature of the system, as well as other non-electrical variables that may be used to describe the state of the memristor. These state variables $\mathbf{x}$ present a dynamic behavior described by g_φ and Eq. 4. As a side comment, it is worthy noticing that all the devices described as being memristors are indeed extended memristors.

When no parasitic effects are present, those extended memristors are better described as generic memristors (or, simply, memristors), since function g_φ depends only on the state variables $\mathbf{x}$ and φ. Ideal memristors, finally, are those corresponding to the
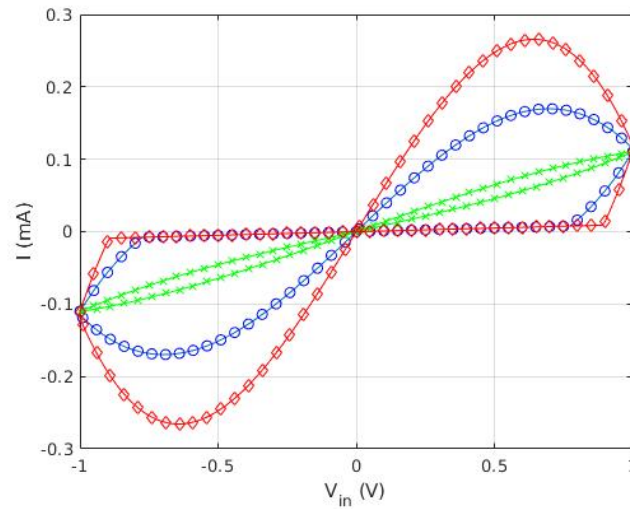
Figure 2. Representation of the I-V characteristics of the memristor defined by Equations 1 and 6 for $G_0 = 0.1\ mS$, $\phi_0 = 10$, and three different frequencies ($\omega(red\ \diamond) < \omega(blue\ o) < \omega(green\ x)$).

original definition[5], and can be considered in this framework as generic memristors with no state variable dependence other than charge or flux.

As an example, let's consider the simplest memristor model that can be devised, similar to those discussed in [24] or, for real devices, in [25–28]. In this case, the model of an ideal memristor where the memresistance or the memconductance depends only on the charge or the flux can be written as:

$$G = G_0\left(1 + \frac{\phi}{\phi_0}\right) \qquad (6)$$

where G_0 is the unperturbed conductance, and ϕ_0 includes the importance of the memristive effect. Notice that for $\phi_0 \to \infty$, the behavior tends to be similar to that of an ideal resistor. The behavior of the device is represented in Fig. 2, for $G_0 = 0.1\ mS$, $\phi_0 = 10$, and three different frequencies. Notice how the device reproduces the two fingerprints of a memristor [7,8]: it presents a pinched loop whose area tends to zero at high frequency (green line, 'x' symbol).

2.2. Cellular Nonlinear Networks

The system discussed in this paper, Celullar Nonlinear Network (CNNs) [1], is not to be confounded with Convolutional Neural Networks (also CNN), even if they share the same acronym. The CNNs discussed here represent a powerful massively parallel, multivariate signal processing paradigm. In their most basic description, they made of independent processing units, called *cells*, where each cell has an input, an output which is feed back as another input, and also feels the effect of the inputs and outputs of its nearest neighbors. These effects are then processed internally into a state variable, and the output is linearly dependent on the result of this processing, with a positive and negative saturation.

As an example of hardware implementation of a CNN, we find [2], where each processing element typically accommodates additional data storage units, which allow the CNNs to store the programming parameters at the cell level. As a result, these Universal Machines (UMs) can be considered as one of the earliest examples of a non-von Neumann computer. Unfortunately, these memory blocks need a large integrated circuit (IC) area to be implemented, which increases significantly the size of each cell. As a consequence, the spatial resolution is quite poor compared with simple image sensors, which is a common problem that CNN-UMs and, as a inherited problem, arrays comprising sensor-processor cells based upon them, suffer from.

Mathematically, the behavior of the i, j cell can be described by a differential equation as:

$$\frac{dx_{ij}}{dt} = -x_{ij} + a_{0,0}f(x_{ij}) + z_{ij} + b_{00}u_{ij}$$
$$+ \sum_{k,i\in N_{i,j}, k\neq i, l\neq j} a_{k-i,l-j}y_{kl}$$
$$+ \sum_{k,i\in N_{i,j}, k\neq i, l\neq j} b_{k-i,l-j}u_{kl} \tag{7}$$

where x_{ij} is the state variable, u_{ij} and y_{ij} are the inputs and outputs, respectively, and a_{ij} and b_{ij} are the *feedback* and *feed forward* coefficients. Function $f(x)$ is a nonlinear function, saturating to a minimum and a maximum values (v_{min} and v_{max}). In this sense, it is similar to the output function of a neuron. However, the most usual shape for it is a piecewise linear function, defined as:

$$f(v) = \frac{1}{2}(|v + v_{sat}| - |v - v_{sat}|) \tag{8}$$

Equation 7 is usually rearranged as:

$$\frac{dx_{ij}}{dt} = g(x_{ij}) + z_{ij} + b_{00}u_{ij}$$
$$+ \sum_{k,l\in N_{i,j}, k\neq i, l\neq} \left(a_{k-i,l-j}y_{kl} + b_{k-i,l-j}u_{kl}\right) \tag{9}$$

$$g(x_{ij}) = -x_{ij} + a_{0,0}f(x_{ij}) \tag{10}$$

Notice that, even if the sum is made over the whole set of integers, we usually restrict ourselves to just the nearest neighbors. This Eq. 10 can be implemented with a single element thanks to the unique non linear behavior of memristors. Thanks to this capability, to process or store data within a common physical nanoscale medium, their use in future CNN cell designs may allow to remove the burden of extra memory blocks within each processing element, allowing the development of co-located sensor-processor arrays with enormous high resolution levels, specially suited for the Internet-of-Things (IoT) industry.

Referred to the determination of the coefficients, some methods have been published [29,30] for the traditional CNN, and , more recently, using the so-called Dynamic Route Map (DRM) [3,4,31,32], to the first-order approximate model of each cell of a Memristor CNN (M-CNN). Such an approximation is depicted as a circuit in Fig. 3, and the evolution of the state variable (x_{ij}) is described by an equation equivalent to Eq. 9:

$$\frac{dx_{ij}}{dt} = k(x_{ij})[\quad \theta(v_{x;i,j}f_{+}^{p}(x_{m;i,j})$$
$$+\theta(-v_{x;i,j}f_{-}^{p}(x_{m;i,j})] \tag{11}$$

where $i \in 1,..., M$, $j \in 1,..., N$, $\theta()$ is the unit step function, $v_{x_{i,j}}$ stands for the voltage across the capacitor C_x, whereas $xm_{i,j}$, and $v_{m_{i,j}} \equiv v_{x_{i,j}}$ denote, respectively, the state and voltage of memristor m_x, whose current is described via the generalized Ohm's law from Eq. 3: $i_{m_{i,j}} = G(x_{m_{i,j}})v_{m_{i,j}}$, with the memductance given by $G(x_{m_{i,j}}) = x_{m_{i,j}}^{-1}$. The non-linear function proposed in [32] to characterize the memristive CNNs are:

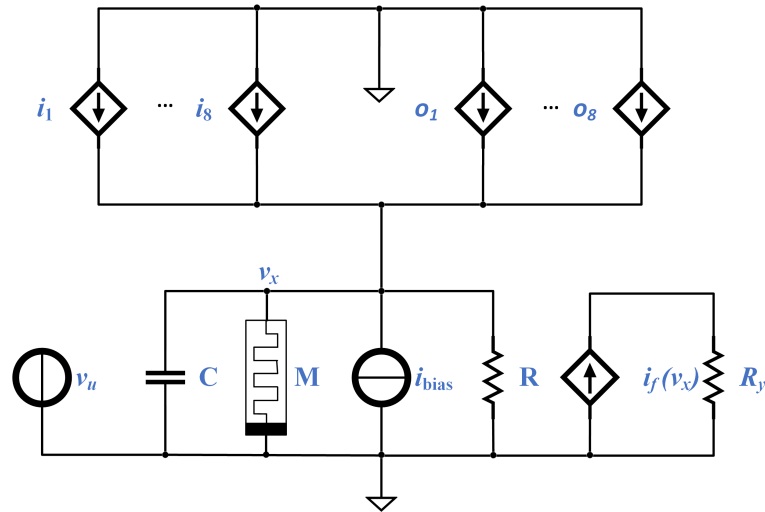$$k(v_{x_{i,j}}) = -\beta v_{x_{i,j}} + \frac{\beta - \alpha}{2}\left(\left|v_{x_{i,j}} + V_t\right| - \left|v_{x_{i,j}} - V_t\right|\right) \tag{12}$$

Figure 3. Schematics and Stochastic implementation of a CNN cell, the processing element in cell C(i,j) (i∈{1,...,M} , j∈{1,...,N}). The other elements (resistor, memristor, and capacitor), present the same values from cell to cell, i.e. Cxi,j=Cx , mxi,j=mx , and Ryi,j=Ry . Adapted from [3].

with $\beta > \alpha \in \Re^+$, featuring units $\Omega V^{-1}s^{-1}$, and $V_t \in \Re^+$ denoting the minimum voltage needed by the memristor for switching. In addition, there are two different window functions, to ensure that the memristor stays in between the two possible states $[x_{on}, x_{off}]$. These two window functions can be written in a compact way [32] as:

$$f_r^p(x) = 1 - \left(\xi + \frac{x - xon}{x_{off} - xon} \right)^{2p} \tag{13}$$

where $\xi = -1$ when $r = "+"$ (the upper boundary), and $\xi = 0$ in the opposite case; p can be any integer ($p \in \mathbb{Z}$). In addition, the dynamical evolution of the voltage $v_{x_{i,j}}$ of each cell in system presented in Fig. 3 is:

$$C_x \frac{dv_{x_{i,j}}}{dt} = -(\frac{1}{R} + G(x_{m_{i,j}})) \cdot v_{x_{i,j}} + a_{0,0} f_{out}(v_{x_{i,j}})$$
$$+ i_{bias} + b_{0,0} v_{in_{j,k}} \tag{14}$$
$$+ \sum_{k,l \in [-1,1]} \left(b_{k,l} v_{in_{i+k,j+l}} + a_{k,l} v_{out_{i+k,k+l}} \right)$$

$$f_{out}(v) = \frac{R_y g_{lin}}{2} (|v + v_{sat}| - |v - v_{sat}|) \tag{15}$$

Notice the equivalence of Eq. 14 with Eq. 9: the last line of both equations is equivalent, while the first line of Eq. 9 corresponds to the first two lines of Eq. 14. Thus, we can conclude that the circuit in Fig. 3 accurately represents a possible implementation of a CNN cell, and it is the circuit we will implement in the next section as a SC module.

3. M-CNN Stochastic Computing
3.1. Stochastic Computing Basics

The SC approach adopts real number representation by strings of N random binary numbers b_i. The probability of "1" bits to appear within the bit-string is proportional to the number to be operated [23]:
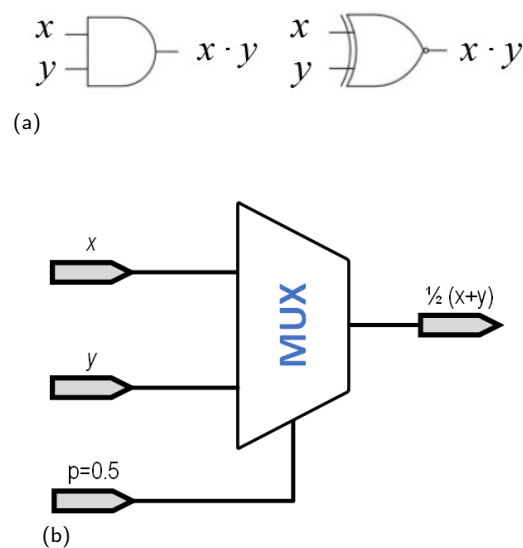
$$p = \frac{1}{N} \sum_i b_i \tag{16}$$

Figure 4. Basic implementation of basic operation in SC. (a) Basic implementation scheme of a SC multiplier in the $(0..1)$ range (AND gate, left) and in the $(-1..1)$ range (XNOR gate, right). (b) Basic implementation scheme of a SC adder using a multiplexer.

These strings are called Stochastic Computing numbers (SCN) or Stochastic Encoded Numbers (SEN). In this paper, we will opt for the second, using also the term Binary Encoded Numbers (BEN) for those encoded as classical binary numbers. Notice that the number of bits that can be encoded in a chain of length N is $log_2 N$, since the relevant information is just the number of "1"s . There are two main ways to generate a map between a SCN and real numbers: first, we can map the desired range of real numbers to the real domain [0..1]; second, we can map them to the interval [-1..1]. Depending on which is mapping to be implemented, many different mathematical operations can then be done using simple logic gates or simple sequential circuits.

As an example, multiplication of SCN is performed using a simple AND gate when using the [0..1] domain. Alternatively, considering the [-1..1] domain, the same multiplicative operation requires the use of an XNOR gate, as shown in Fig. 4a.

Since we cannot represent any SC number as a probability higher than one, the case of addition becomes slightly more complex, since 1+1=2. Thus, the operation that should be implemented is $(x+y)/2$, which would always return a maximum value of 1. This operation is usually implemented using a multiplexer, as shown in Fig. 4b, where the p(0.5) means a signal with a probability of 50% to be '1' or '0'. This necessary input signal is generated using one of the bits generated in the RNG, so no additional circuitry is needed. It is worth pointing out that this gate is the same in both the [0..1] and the [-1..1] domains. Other more complex operations (division[17], square roots[17], reversible gates[33], etc...) are also discussed in the literature, though not presented in this paper.

Another important point is the conversion from BEN to SEN. This is usually achieved by using a scheme similar to that in Fig. 5, where an N-bit random number is generated by utilizing a random number generator (RNG) and compared to the value of the N-bit BEN. If the output of the RNG is below the BEN, the converter's output would be bit "1", bit "0" otherwise. In the opposite operation, converting SEN back to its BEN representation, the number of 1's included in the signal needs to be calculated; something that can be straightforwardly achieved by a simple counter.

It is apparent that the error in the approximation of the SEN to its actual value is equivalent to the error provided by a random walk process of length n, and thus proportional to $\sqrt{n}$, as it has been discussed in the literature [34]. Therefore, using N bits, we may consider that all the noise caused by the process is included in the lowest
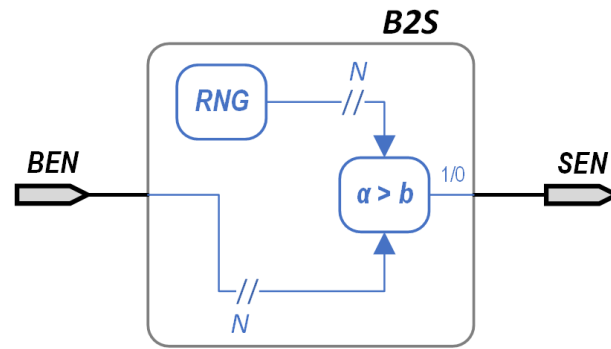
Figure 5. Basic implementation scheme of a Binary Encoded Number (BEN) to a Stochastic Encoded Number (SEN), using a Random Number Generator (RNG).
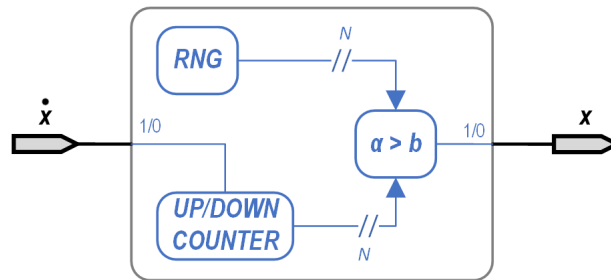

Figure 6. Basic implementation scheme of a SC integrator. Notice that both the input $\dot{x}(t)$ and the output $x(t)$ are SEN numbers.

$N/2$ bits. This way, the noise figure NF for a signal of power S_p with noise power N_p caused by the use of the SEN is:

$$NF = 10log_{10}\left(\frac{S_p}{N_p}\right) = 10log_{10}\left(\frac{2^N}{2^{N/2}}\right) \approx 3.01N/2 \; dB \tag{17}$$

Notice that for this equation we have considered the maximum possible amplitude for the input signal. In order to consider the minimum amplitude *over* the noise, we consider we are 1 bit over the noise $(N/2 + 1)$. In this case, we obtain:

$$NF = 10log_{10}\left(\frac{S_p}{N_p}\right) = 10log_{10}\left(\frac{2^{N/2+1}}{2^{N/2}}\right) \approx 3.01 \; dB \tag{18}$$

Thus, the system is expected to have a NF between 3dB and $3\sqrt{N}$dB. This desired NF would set the required number of bits, which is related to the sensitivity of the equation system, on noise. Empirically, we have seen that linear equations allow for a low N, while nonlinear systems call for higher values. Notice that a value of the $NF = 20$dB calls for N=12 bits, while for N=32 bits the provided noise figure $NF = 54$dB.

3.2. Stochastic Computing Implementation of a Memristor Emulator

The described advantages of SC framework were used by a stochastic computing implementation of a memristor emulator [9], which describes the memristor using equation (6), and was written in the form of equations (19)-(21), so that it could be implemented in a discrete way. The mathematical operations eqs. (19)-(21) were carried out by simple digital gates. A simple manner to model memristors is using a linear relation of the charge Q with the memconductance $G(Q)$, with an upper and a lower value, G_{max} and G_{min}, correspondingly. Then according to this approach:

$$G_1(Q) = G_0 + G_1 \cdot Q \tag{19}$$
$$G_2(Q) = min(G_{max}, G_1(Q)) \tag{20}$$
$$G(Q) = max(G_{min}, G_2(Q)) \tag{21}$$

It is apparent that now Stochastic Computing can be used to implement such a model within a digital environment (an FPGA, or an ASIC). Other complex, physically-based models simulating memristive behavior in FPGAs can be found in the literature [35,36], but they are very mathematically complex models, requiring a very large number of gates. The emulator discussed above presents all the standard fingerprints of a memristor as required by the theory [8]. This implementation appears as a block diagram in Fig. 7, where v_x and GND are the SCN values of the positive and negative terminals of the memristor, respectively, while the calculated current is represented by the stochastic value i_M. The SCN value of GND has to be represented by a probability of 0.5 for an "1", since we are mapping the interval [0..1] to an interval that includes negative and positive values.

Rewriting equations (19)-(21) to allow them to be implemented in discrete time results in:

$$Q = \int^t i(t) \approx \Delta t \cdot \sum_j i(t = j \cdot \Delta t) \tag{22}$$

where the integration step is Δt. Using Eq. (22), we can rewrite (19) as:

$$G_1(Q) = G_0 + G_1 \cdot \Delta t \cdot \sum_j i(t = j \cdot \Delta t)S \tag{23}$$

The adder needs to increase or decrease its output by a unit, depending on the inputs, since i and GND are both SEN. An increasing is performed when $v_x = 1$, while when $GND = 1$ the output is reduced. A single constant is used to groupe M_1 and Δt, and the *max* and *min* functions are built into the adder by establishing a maximum and a minimum values. A SEN output is generated from the adder's output by comparing
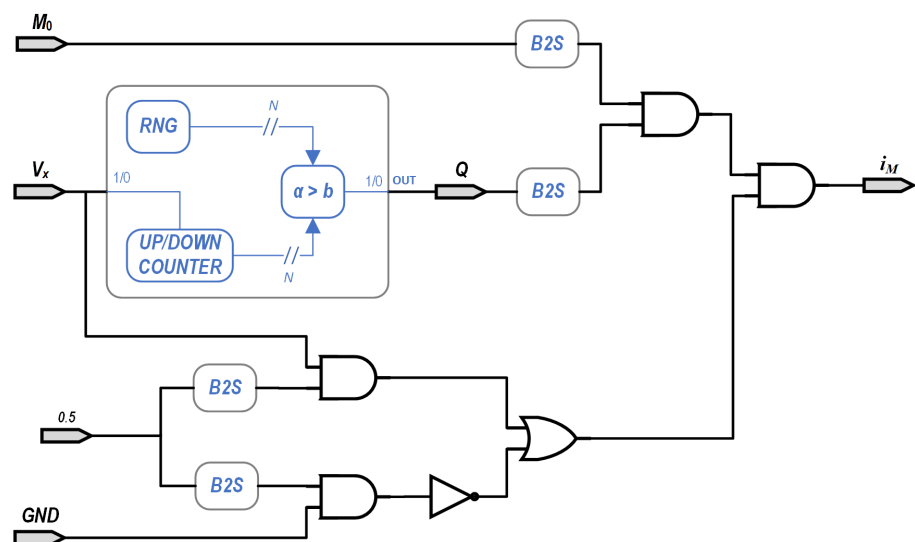

Figure 7. The Stochastic Computing memristor implementation. Inputs v_x and GND are the SCN values of the positive and negative inputs of the memristor respectively, and i_M is the calculated SCN value of the current.
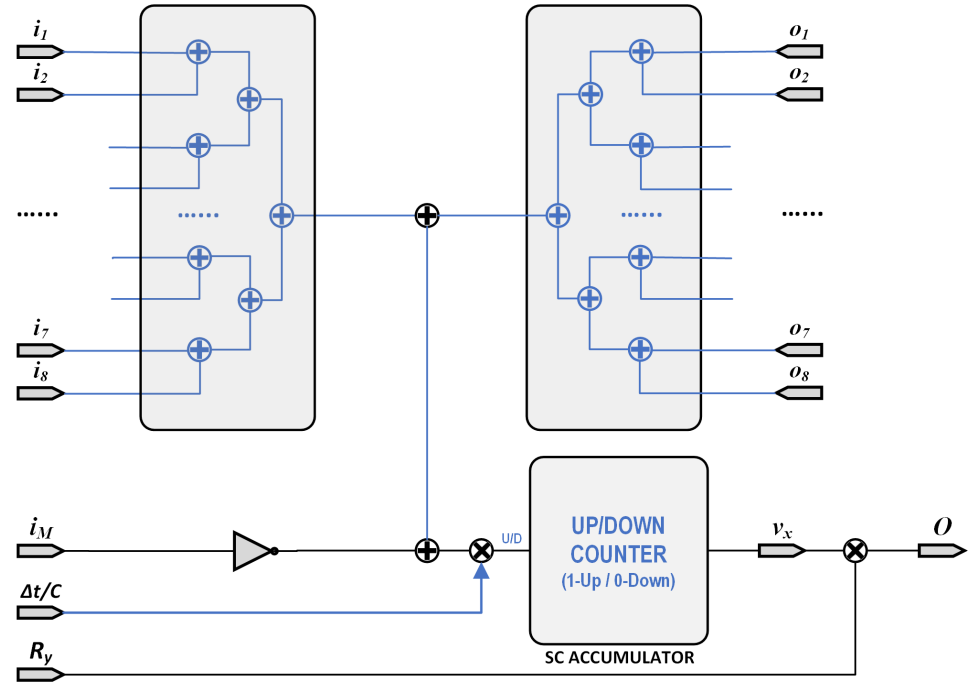
Figure 8. Stochastic Computing Circuit implementation of the M-CNN processing element C(i,j) as in Fig. 3.

the it to a random number spanning $[0..(2^{NB} - 1)]$. An AND gate is then used to obtain the current as per Eq. (1), as in Fig. 7.

3.3. Stochastic Computing implementation of a M-CNN

As discussed above, a very compact implementation of a single cell of a Cellular Nonlinear Network single cells can be performed as in Fig. 3 [3]. The output voltage $v_{y;i,j}$ presents a non-linear dependence on the internal voltage $v_{x;i,j}$. The dynamic behavior of this v_x is governed by a differential equation:

$$\frac{dv_x}{dt} = \frac{1}{C}\left(\sum_{i,j}(B_{i,j}i_{i,j} + A_{i,j}o_{i,j}) - i_M\right) \tag{24}$$

where i_i are the input currents caused by the *inputs* of the nearest cells. The currents o_i correspond to the *outputs* of those cells, while the current i_M is that flowing through the memristor. As proposed in [3,4], we only consider the 8 closest neighbors. This way, Stochastic Computing can be used to make an implementation of this equation. As an initial step, we do a first order integration of 24:

$$\Delta v_x = \frac{\Delta t}{C}\left(\sum_i(B_i i_i + A_i o_i) - i_M\right) \tag{25}$$

The above Eq. (25) has been implemented in Fig. 8 as a stochastic equivalent circuit, where all data circulation correspond to 1-bit lines. Thus, implementation of Eq. 25 has been performed with 16 adders each implemented using 1 *OR* gate with a 1-bit *multiplexers*, 2 multipliers implemented as *AND* gates, 1 inverter, and 1 accumulator. Additionally, a random number generator is also needed (note that it may be shared between different cells) along with the memristor emulator previously presented.

The speed of the system can be estimated with the length of the chain needed to implement the SCN representation of the numbers to be used. These numbers are determined considering the input images. In our case, we are using both gray images and color images. The gray images use a 8-bit single plane to store it, while the color

images use three different 8-bit planes. Thus, at least 8 bits need to be recovered faithfully. As has been discussed above [34], the use of 14 bits corresponds to a chain length of $2^{14} = 16384$ bits and can represent values with to an error confined in the last 6 bits, with more than a 95% probability. Following this reasoning, we choose these 14 bits for both the length of the chain and the accumulators.

4. Image Processing Results

The CNN described above was simulated using Matlab, with the circuit parameters equivalent to those appearing in Table 1. The FPGA-in-the-loop methodology, as shown in Fig. 1, was implemented to speed up the simulation, using an Arria V development kit. This FPGA system was connected to the computer running the Matlab code via a cabled network.

Table 1: Parameter values for the elements of the circuit in Fig. 3, where the memristor is defined by Eq. 1

Parameter	Value
R	100 kΩ
C	50 μF
G_0	100 kΩ
ϕ_0	10 V·s

The A and B matrices were changed to three different sets, corresponding to three different cases: store, edge, sharpening, as discussed below. The notation to represent the coefficients in the matrices is shown in Table 2.

Table 2: Coefficient notation for $M = A, B$ ($m = a, b$). Notice that the coefficient for the current node is $(0,0)$.

$$M \quad \begin{pmatrix} m_{-1,-1} & m_{0,-1} & m_{1,-1} \\ m_{-1,0} & m_{0,0} & m_{1,0} \\ m_{-1,1} & m_{0,1} & m_{1,1} \end{pmatrix}$$

The images have all different size (notated as NxM), as reported in Table 3, and they were all initially color images with 8-bit color resolution at each RGB plane. We have processed these figures by performing two different experiments, both of them using three different (A,B) sets of parameters: first, we used a color-to-gray conversion, and we processed the images through the three different CNN. In a second step, we used a color image, and we processed each color plane independently to finally reconstruct a color image from these three planes.

Table 3: Picture size in pixels. The color depth is 8 bits per channel.

Figure	Size (M x N) pixels^2
Fig. 9	512 x 512
Fig. 10	768 x 512
Fig. 12	768 x 512

In order to keep a good NF during the stochastic processing, and according to Eq. 17 and the discussion in the previous section, each pixel was converted from 8 to 14-bits by padding the least significant positions with zeros. After the processing, the stochastic images to normal images were converted back by disregarding the 6 least significant bits of the corresponding accumulator in Fig. 7.

The results are shown in two different ways: as pictures, and also using the RMS error and the entropy of the image. The RMS error e_{rms} is defined in Eq. 26, where

$p_{i;i,j}$ and $p_{o;i,j}$ are the values of the pixel (i, j) for the input and the processed image, respectively. The entropy H is calculated using the Matlab implementation of Eq. 27, where p_i contains the normalized histogram counts for each gray level. The entropy of the unprocessed images is reported in Table 4. Notice that for the color images we report the values of the entropy for each channel, while for the rest we report only the entropy of the gray image.

$$e_{rms} = \frac{1}{N \cdot M} \sum_{i,j} (p_{o;i,j} - p_{p;i,j})^2 \tag{26}$$

$$H = -\sum_i p_i log_2(p_i) \tag{27}$$

Table 4: Calculated values of entropy of the images. The results for the color image show the three color planes separately.

Figure	Original	Store	Edge	Enhance
Fig. 9	6.70	6.71	4.03	6.90
Fig. 10	7.18	7.20	3.87	7.45
Fig. 12 (R)	7.15	7.07	4.04	7.38
Fig. 12 (G)	7.16	7.27	4.19	7.88
Fig. 12 (B)	7.16	7.28	4.26	7.76

4.1. Store image

As a first example, we show the results of storing the image into the CNN. That is, the values of internal voltage are evolved until the output is equal to the input. As a comment, this is the easiest "program" that can be implemented into the SM-CNN, and can be used as a first step for more complex algorithms as can be, for instance, a background removal or a motion detection algorithms. The coefficients of the matrices are provided in Table 5.

Table 5: Coefficients for the input and output weights in Eq. 24 for the case of the image store setup.

$$A \quad \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.9 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

$$B \quad \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

We have represented both the input and output images for two different cases input gray images (Figures 9a and 10a) and a color image (Fig. 12a). The results for the store process are shown in Figures 9b and 10b for the gray images, while the stored color image is presented in Fig. 12b. Visually, it can be seen there that the algorithm performs correctly.

Additionally, we have calculated the entropy of the images and the RMS error, as shown in Tables 4 and 6. The entropy of the images is nearly the same, and the rms error is kept, at most, below 1.3%.

(a) Original image.

(b) Stored image.

(c) Edge detection result.

(d) Sharpened image.

Figure 9. Example 1: results obtained using the three proposed stochastic computing CNN with different gene values.

Table 6: Calculated RMS of the stored images, referred to the original image. The results for the color image show the three color planes separately.

Figure	RMS
Fig. 9b	1.29%
Fig. 10b	0.57%
Fig. 12b (R)	1.13%
Fig. 12b (G)	0.58%
Fig. 12b (B)	0.61%

4.2. EDGE detection using SM-CNN

The proposed SM-CNN was further checked using an implementation of one of the stochastic systems proposed in [4]. Specifically, we have improved the EDGE routine presented in [10]. This routine performs a border detection algorithm in the image
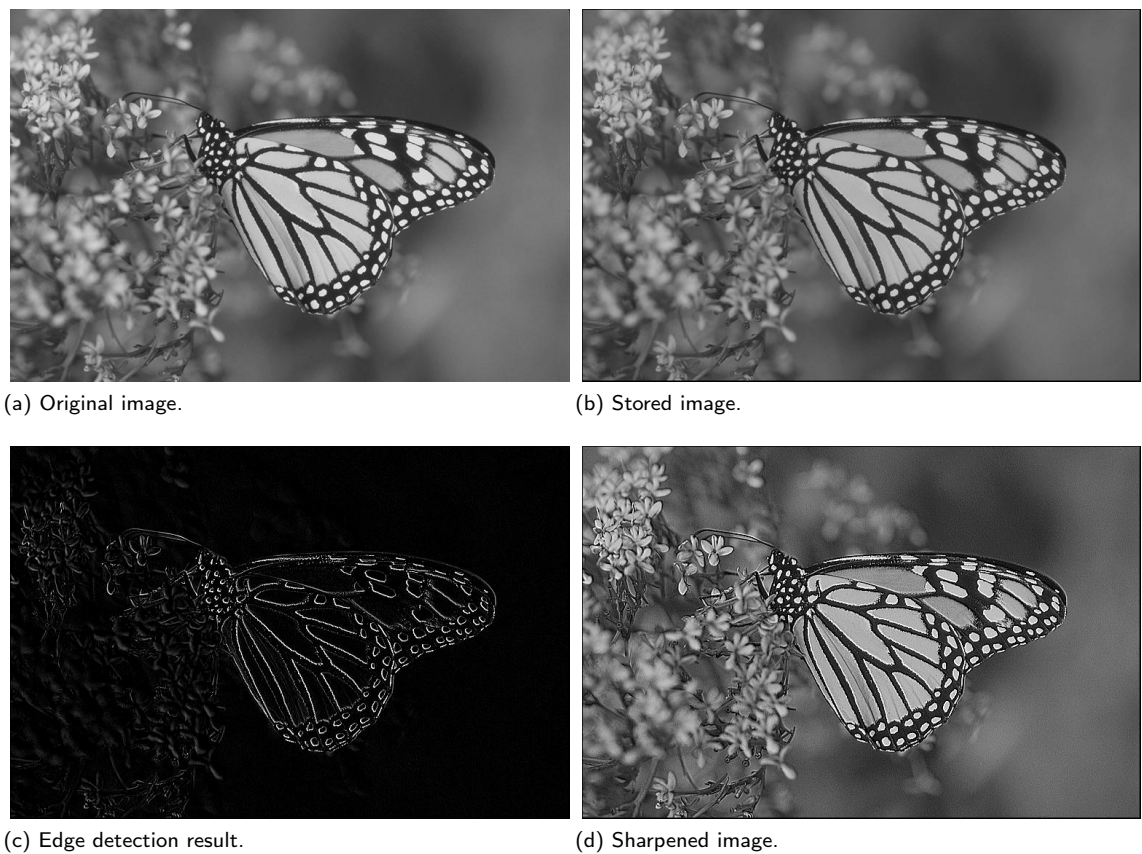
(a) Original image.   (b) Stored image.

(c) Edge detection result.   (d) Sharpened image.

Figure 10. Example 2: results obtained using the three proposed stochastic computing CNN with different gene values.

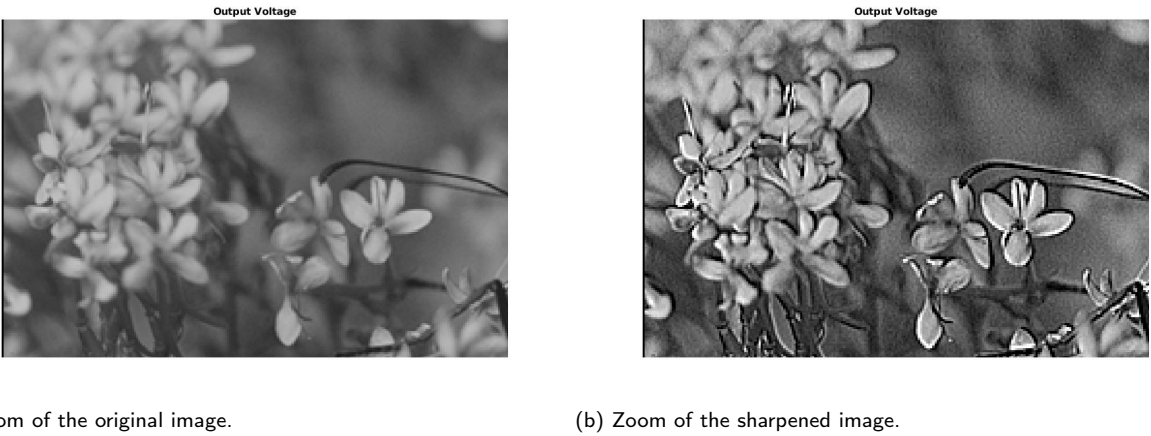(a) Zoom of the original image.   (b) Zoom of the sharpened image.

Figure 11. Example 2: zoom in of Fig 10a and 10d, showing a detail of the sharpening results obtained using the proposed stochastic computing CNN.

using the coefficients in Table 7. In the previous work, the routine was fixed, and no quantitative analysis was performed. The edge algorithm aims to detect changes between adjacent pixels, so the output value will evolve to a 1 or 0, depending on the change of color. The evolution will depend on the threshold of the output function and can thus be changed.

We have used the same images as in the previous example, where two of the images were gray 8-bit images and another one was a 24-bit color image (3x8 bits planes). They were processed as in the previous case to 14-bits by padding, and back to 8-bits by truncation. We have represented both the input and output images for two different gray images in Fig. 9c and Fig. 10c, while the result for the color image is shown in

(a) Original image.

(b) Stored image.

(c) Edge detection result.

(d) Sharpened image.

Figure 12. Example 3: Color figure, showing the sharpening results obtained using the proposed stochastic computing CNN.

(a) Zoom of the original image.
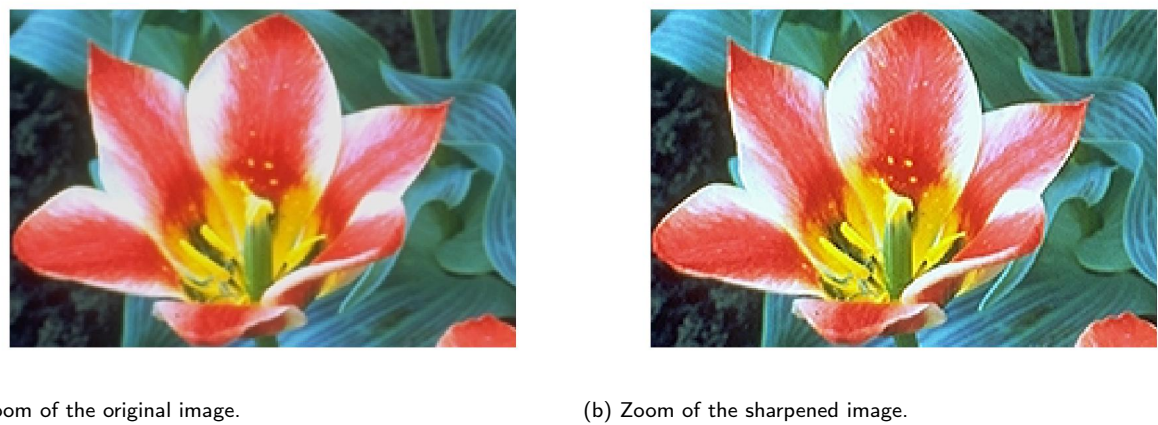
(b) Zoom of the sharpened image.

Figure 13. Example 3: zoom in of Fig 12a and 12d, showing a detail of the sharpening results obtained using the proposed stochastic computing CNN.

Fig. 12c. It can be seen there that the algorithm performs as expected, showing also the corresponding decreasing in the values of the entropy in Table 4.

4.3. Sharpening

The sharpening algorithm is a variation of the EDGE detection, combined with the STORE genii. In fact, we have calculated a new coefficient set as a linear combination

Table 7: Coefficients for the input and output weights in Eq. 24 for the case of the edge detection setup.

A	$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$
B	$\begin{pmatrix} -1/8 & -1/8 & -1/8 \\ -1/8 & 1.0 & -1/8 \\ -1/8 & -1/8 & -1/8 \end{pmatrix}$

of the two previous sets: A_s, B_s for the store matrices, and A_e, B_e for the edge detection. The new set A_i, B_i is calculated as:

$$M_i = \lambda_1 M_s + \lambda_2 M_i \tag{28}$$

where M stands for both A and B. In this case, $\lambda_1 = 1/3$ and $\lambda_2 = 2/5$, and the corresponding matrix coefficients are provided in Table 8.

Table 8: Coefficients for the input and output weights in Eq. 24 for the case of the image enhancement setup.

A	$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$
B	$\begin{pmatrix} -0.05 & -0.05 & -0.05 \\ -0.05 & 0.43 & -0.05 \\ -0.05 & -0.05 & -0.05 \end{pmatrix}$

    We have used the same images as in the previous example, where two of the images were gray 8-bit images and another one was a 24-bit color image (3x8 bits planes). They were processed as in the previous case to 14-bits by padding, and back to 8-bits by truncation. Results of applying this set of coefficients to gray images are shown in Fig. 9d, 10d (with a zoom comparing original in Fig. 11a against the processed output in 11b). The results for a color image are depicted in Fig. 12d, with a zoom in shown in Fig. 13a and 13b for the original and processed images, respectively. Visually, the images seem to be improved, with less fuzzy edges. This is further corroborated by the increase in the entropy shown in Table 4.

5. Conclusion

    In this work we have performed a fully digital implementation of a Memristive Cellular Nonlinear Network profiting from the Stochastic Computing paradigm. The basic unitary cell of the proposed CNN features a digital memristor emulator, plus several arithmetic units that are implemented as very simple gates, allowing for an enormous number of cells in parallel, which can translate into a very fast image processor.

    We have implemented this full CNN structure into an ARRIA V FPGA, and we have tested it along with Matlab by implementing three different procedures: a image store, an edge detection, and, finally, an image sharpening process. Notice that these three procedures involve only the change of the matrix coefficients, that are common to all the cells. As has been discussed, the results imply that the system performs smoothly, with errors lower than 1.3% in the storage, an excellent edge detection capability, and a very

good detail sharpening. A full FPGA implementation of images with lower number of pixels would allow for a very high image processing speed, adequate for real time needs, and well inside the capabilities and requirements of edge computing. Notice that in this paper only the proof of concept for the SC Memristive CNN has been discussed, not comparing it against any other improving algorithm using, for instance, a hard-wired algorithm implementation or Neural Networks, which may show much better image improvement. It has to be noted, however, that the method presented here is training-free, which simplifies the design when compared against NNs and also removes any possible bias introduced by the training. In addition, the facility to change the algorithm is also worth to mention, since it reduces to changing the values of the coefficients in the the cells. This makes this approach specially suited over a hard implementation of specific algorithms in, for instance, multi-purpose systems that can need to be swapping functions on the fly.

1. Chua, L.O.; Yang, L. Cellular neural networks: Theory. *IEEE Transactions on circuits and systems* **1988**, *35*, 1257–1272.
2. Rodriguez-Vazquez, A.; Fernandez-Berni, J.; Lenero-Bardallo, J.A.; Vornicu, I.; Carmona-Galan, R. CMOS vision sensors: embedding computer vision at imaging front-ends. *IEEE Circuits and Systems Magazine* **2018**, *18*, 90–107.
3. Tetzlaff, R.; Ascoli, A.; Messaris, I.; Chua, L.O. Theoretical Foundations of Memristor Cellular Nonlinear Networks: Memcomputing With Bistable-Like Memristors. *IEEE Transactions on Circuits and Systems I: Regular Papers* **2020**, *67*, 502–515.
4. Ascoli, A.; Tetzlaff, R.; Kang, S.; Chua, L.O. Theoretical Foundations of Memristor Cellular Nonlinear Networks: A DRM$_2$-Based Method to Design Memcomputers With Dynamic Memristors. *IEEE Transactions on Circuits and Systems I: Regular Papers* **2020**, pp. 1–14.
5. Chua, L.O. Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on* **1971**, *18*, 507–519.
6. Strukov, D.B.; Snider, G.S.; Stewart, D.R.; Williams, R.S. The missing memristor found. *nature* **2008**, *453*, 80–83.
7. Chua, L.O. Everything you wish to know about memristors but are afraid to ask. *Radioengineering* **2015**, *24*, 319.
8. Biolek, D.; Biolek, Z.; Biolková, V.; Kolka, Z. Some fingerprints of ideal memristors. Circuits and Systems (ISCAS), 2013 IEEE International Symposium on. IEEE, 2013, pp. 201–204.
9. Camps, O.; Picos, R.; de Benito, C.; Chawa, M.M.A.; Stavrinides, S.G. Emulating memristors in a digital environment using stochastic logic. 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), 2018, pp. 1–4.
10. Camps, O.; Stavrinides, S.G.; Picos, R. Efficient Implementation of Memristor Cellular Nonlinear Networks using Stochastic Computing. 2020 European Conference on Circuit Theory and Design (ECCTD). IEEE, 2020, pp. 1–4.
11. Von Neumann, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies* **1956**, *34*, 43–98.

12. Gaines, B.R. Stochastic computing systems. In *Advances in information systems science*; Springer, 1969; pp. 37–172.

13. Wang, R.; Han, J.; Cockburn, B.; Elliott, D. Stochastic circuit design and performance evaluation of vector quantization. Application-specific Systems, Architectures and Processors (ASAP) IEEE 26th Int. Conf. on. IEEE, 2015, pp. 111–115.

14. Camps, O.; Stavrinides, S.G.; Picos, R. Stochastic Computing Implementation of Chaotic Systems. *Mathematics* **2021**, *9*, 375.

15. Morro, A.; Canals, V.; Oliver, A.; Alomar, M.L.; Rossello, J.L. Ultra-fast data-mining hardware architecture based on stochastic computing. *PloS one* **2015**, *10*, e0124176.

16. Yuan, B.; Wang, Y.; Wang, Z. Area-efficient scaling-free DFT/FFT design using stochastic computing. *IEEE Transactions on Circuits and Systems II: Express Briefs* **2016**, *63*, 1131–1135.

17. Marin, S.T.; Reboul, J.Q.; Franquelo, L.G. Digital stochastic realization of complex analog controllers. *IEEE Transactions on Industrial Electronics* **2002**, *49*, 1101–1109.

18. Najafi, M.H.; Jamali-Zavareh, S.; Lilja, D.J.; Riedel, M.D.; Bazargan, K.; Harjani, R. Time-encoded values for highly efficient stochastic circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2017**, *25*, 1644–1657.

19. Fick, D.; Kim, G.; Wang, A.; Blaauw, D.; Sylvester, D. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering. Proceedings of the IEEE 2014 Custom Integrated Circuits Conference. IEEE, 2014, pp. 1–4.

20. Toral, S.; Quero, J.; Ortega, J.; Franquelo, L. Stochastic A/D sigma-delta converter on FPGA. Circuits and Systems, 1999. 42nd Midwest Symposium on. IEEE, 1999, Vol. 1, pp. 35–38.

21. Moons, B.; Verhelst, M. Energy-Efficiency and Accuracy of Stochastic Computing Circuits in Emerging Technologies. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **2014**, *4*, 475–486. doi:10.1109/JETCAS.2014.2361070.

22. Li, S.; Glova, A.O.; Hu, X.; Gu, P.; Niu, D.; Malladi, K.T.; Zheng, H.; Brennan, B.; Xie, Y. SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator. MICRO, 2018, pp. 696–709.

23. Toral, S.; Quero, J.; Franquelo, L. Stochastic pulse coded arithmetic. Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on. IEEE, 2000, Vol. 1, pp. 599–602.

24. Corinto, F.; Civalleri, P.P.; Chua, L.O. A theoretical approach to memristor devices **2015**.

25. Al Chawa, M.; Picos, R.; Covi, E.; Brivio, S.; Garcia-Moreno, E.; Spiga, S. Flux-charge characterizing of reset transition in bipolar resistive-switching memristive devices. Proc. 11th Spanish Conf. Electron Devices, 2017.

26. Picos, R.; Roldan, J.B.; Al Chawa, M.M.; Garcia-Fernandez, P.; Jimenez-Molinos, F.; Garcia-Moreno, E. Semiempirical Modeling of Reset Transitions in Unipolar Resistive-Switching Based Memristors. *RADIOENGINEERING* **2015**, *24*, 421.

27. Picos, R.; Roldan, J.; Al Chawa, M.; Jimenez-Molinos, F.; Villena, M.; Garcia-Moreno, E. Exploring ReRAM-based memristors in the charge-flux domain, a modeling approach. Memristive Systems (MEMRISYS) 2015 International Conference on. IEEE, 2015, pp. 1–2.

28. Al Chawa, M.M.; Picos, R. A Simple Quasi-Static Compact Model of Bipolar ReRAM Memristive Devices. *IEEE Trans. on Circuits and Systems II: Express Briefs* **2019**.

29. Itoh, M.; Chua, L.O. Designing CNN genes. *International Journal of Bifurcation and Chaos* **2003**, *13*, 2739–2824.

30. Itoh, M. Some Interesting Features of Memristor CNN. *arXiv preprint arXiv:1902.05167* **2019**.

31. Maldonado, D.; Gonzalez, M.B.; Campabadal, F.; Jimenez-Molinos, F.; Al Chawa, M.M.; Stavrinides, S.G.; Roldan, J.B.; Tetzlaff, R.; Picos, R.; Chua, L.O. Experimental evaluation of the dynamic route map in the reset transition of memristive ReRAMs. *Chaos, Solitons & Fractals* **2020**, *139*, 110288.

32. Ascoli, A.; Tetzlaff, R.; Messaris, I.; Kang, S.; Chua, L. Image Processing by Cellular Mem-computing Structures. 2020 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2020, pp. 1–5.

33. Khanday, F.A.; Akhtar, R. Reversible stochastic computing. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields* **2020**, *n/a*, e2711, [https://onlinelibrary.wiley.com/doi/pdf/10.1002/jnm.2711]. doi:10.1002/jnm.2711.

34. Camps, O.; Picos, R.; de Benito, C.; Al Chawa, M.M.; Stavrinides, S.G. Effective accuracy estimation and representation error reduction for stochastic logic operations. 2018 7th Int. Conf. on Modern Circuits and Systems Technologies (MOCAST). IEEE, 2018, pp. 1–4.

35. Vourkas, I.; Abusleme, A.; Ntinas, V.; Sirakoulis, G.C.; Rubio, A. A Digital Memristor Emulator for FPGA-Based Artificial Neural Networks. Verification and Security Workshop (IVSW), IEEE International. IEEE, 2016, pp. 1–4.

36. Ranjan, R.; Ponce, P.M.; Kankuppe, A.; John, B.; Saleh, L.A.; Schroeder, D.; Krautschneider, W.H. Programmable memristor emulator asic for biologically inspired memristive learning. Telecommunications and Signal Processing (TSP), 2016 39th International Conference on. IEEE, 2016, pp. 261–264.