mathematics

MDPI

Article

Implementation of the Hindmarsh–Rose Model Using Stochastic Computing

Oscar Camps [1], Stavros G. Stavrinides [2], Carol de Benito [1,3] and Rodrigo Picos [1,3,*]

[1] Industrial Engineering and Construction Department, University of Balearic Islands, 07122, Palma, Spain; oscar.camps.pascual@gmail.com
[2] Physics Department, International Hellenic University, 65404, Kavala, Greece; s.stavrinides@ihu.edu.gr
[3] Balearic Islands Health Institute (IdISBa) 07120, Palma, Spain
* Correspondence: rodrigo.picos@uib.es

Abstract: The Hindmarsh-Rose model is one of the most used models to reproduce spiking behaviour in biological neurons. However, since it is defined as a system of three coupled differential equations, its implementation can be burdensome, and impractical for a large number of elements. In this paper we present a successful implementation of this model within a stochastic computing environment. The merits of the proposed approach are design simplicity, due to stochastic computing, and the ease of implementation. Simulation results showed that the approximation achieved is equivalent to introducing a noise source into the original model, in order to reproduce actual observed behaviour of biological systems. A study for the level of noise introduced, according to the number of bits in the stochastic sequence, has been performed. Additionally, we demonstrate that such an approach, even though it is noisy, reproduces the behaviour of biological systems, which are intrinsically noisy. It is also demonstrated that using some 18-19 bits are enough to provide a speedup of x2 compared to biological systems, with a very small number of gates, thus paving the road for the *in silico* implementation of large neuron networks.

Keywords: stochastic logic; chaotic systems; approximate computing; Hindmarsh Rose system

check for updates

Citation: Camps, O.; Stavrinides, S.G.; de Benito, C.; Picos, R. Implementation of the Hindmarsh–Rose model Using Stochastic Computing. *Mathematics* **2021**, *1*, 5. https://doi.org/

Received:
Accepted:
Published:

1. Introduction

Simulation of the brain is one of the big issues for this century. The initial simulations were performed using models for single neurons [1], but are currently focusing on multiscale modelling [2] in order to properly address the incredible complexity of the human brain. The problems outlined in the literature can be divided into two different broad categories: theoretical and implementation-related ones. The theoretical problems deal with the different problems on how to model the neurons, the connections, and the structures inside the brain. On the other hand, the current proposals for implementing the required infrastructure cover many different approaches, including the actual physical devices, but also the architecture.

Related to the physical devices where the simulation is performed, the literature includes proposals ranging from quantum computing [3], to the use of memristors to implement biologically inspired neural networks [4] or to model specific parts as the synapses [5] , multi-scale hardware simulation of the brains [6] , or accelerator for neural network simulation using analog elements [7]. The architectures implementing full models use neural networks [8], simulation at the level of tissues [9], interactions with different parts [10], or descriptions of the communications at a higher level [11].

Another main problem for simulating the human brain related to implementation, is the one of scaling: the human brain has around 86 billion neurons, with thousands of inputs each. Thus, one of the biggest challenges is the required computing infrastructure. There are many different activities related to this, usually based on high performance computing resources, requiring huge amounts of power. Among others, we can cite some:

the European Human Brain project[12] which is a 10-year long European funded project, that has lead to the creation of the Virtual Brain service in EBRAINS[13]; the Open Source Brain [14] which provides a collection of open source tools for simulating and visualizing specific models and parts of the brain; and the Swiss Blue Brain Nexus, which aims to implement and simulate the brain of a mouse [15].

Considering the above, approximate computing emerges as an important technological pathway, since it enables high power and resource savings [16]. The price is that this framework trades accuracy for savings. To this direction, some methods for successfully implementing approximate computing, ranging from software-based approaches (programming methods and algorithms) to hardware implementations or ubiquitous solutions, have been proposed.

One of such approaches is Stochastic Computing (SC), proposed by Von Neumann in 1956 [17], making a trade off between calculation time and accuracy. It offers significant advantages, the most important of them being the utilization of simple logic operations and the significant reduction of circuit components, when we refer to hardware implementation, apparently, reducing the power consumption. The main drawback of this approach is the increased logic-operation time demanded, which in its turn increases the total power consumption when the number of bits exceeds 16–17 [18,19]; but there have already appeared techniques that allow this problem to be alleviated, making SC competitive even for higher bit-numbers [19].It has to be mentioned that the property of stochasticity demanded for number generation could be considered as an equivalent of noisy system, its existence being many times beneficial for the designed system. It should be noted that neuromorphic computing as well as biological neurons themselves demonstrate characteristics that lay very close to stochastic computing ones, leading to a noteworthy implementation compatibility, i.e., the sets of consecutive spikes in neuromorphic computing or the neurons, as these are compared to the pulse-trains in stochastic computing [20].

As mentioned above, one of the problems for simulating large sets of biological neurons is the large amount of computational resources needed to simulate each neuron. Since the SC approach allows for implementing large sets of neurons at a relatively low cost of resources, we will study in this paper the possibility of successfully implementing the Hindmarsh–Rose (HR) model within a SC environment. The merits of the proposed approach are design simplicity (due to SC) and the ease of implementation, requiring a very few number of gates compared to the number required for a classical arithmetic implementation. Simulation results in Matlab showed that the SC achieved approximation is equivalent to introducing a noise source into the original model. Consequently, a study for the level of noise introduced, according to the number of bits in the stochastic sequence, has been performed. Additionally, we demonstrate that such an approach, even though it is noisy, it is close to a real biological neuron, since noise is an intrinsic characteristic playing a very significant role.

The rest of the paper is organized to follow this goal. The next section introduces the main concepts of stochastic computing, in order to provide a minimum basis on this issue. Section 3 introduces the Hindmarsh-Rose model, and how we have implemented it in SC. Section 4 presents results obtained using different numbers of bits, as well as studying the variability between different runs. Also in this section, the equivalent noise level is studied and characterized in terms of the number of equivalent bits. Finally, the last section discusses the results and provides some hindsight on possible applications.

2. Stochastic Computing Implementation of Analog Systems

2.1. *Stochastic Computing Basics*

In the approach proposed by Stochastic Computing (SC), numbers are ideally represented by probabilities [17,21,22]. Thus, operations between numbers can be represented as compound probabilities. However, and due to the fact that we cannot represent an actual probability p, we estimate this probability as the average value of a set of samples P, thus assuming $p \approx < P >$ [23] and also obtaining an error for the estimation ϵ_p, that can be

Table 1. Implementation of basic operations in Stochastic Computing in the [−1,1] range. The value of p is such that it is 0 or 1 with a 0.5 probability, $\bar{p} = 1 - p$, and q is a random variable with a normal distribution between -1 and 1. Notice that these functions can be trivially expanded to work for arbitrary vector length.

Operation	Implementation	Function Name
(x+y)/2	OR(AND(p,x), AND($\bar{p}$,y))	add(x,y)
x*y	XNOR(x,y)	mult(x,y)
-x	NOT(x)	neg(x)
Real number to SEN	0 if q>x; 1 otherwise	get_sn(x)

related to the standard deviation σ_p of the average value p. In SC, the set of samples is represented by a string of binary values, and is usually referred to as Stochastic Computing numbers (SCN) or Stochastic Encoded Numbers (SEN). As for the rest of this paper, we will use the second calling convention, as well as also using Binary Encoded Numbers (BEN) as the name for those numbers that are encoded as classical binary numbers. It has to be noted that two main mappings from real to SEN are typically used, either from the real domain [0,1], or from the real domain [−1,1].

The most basic operations to be performed in SC are multiplication and addition, and their implementation depends on the selected mapping. In our case, using the [−1,1] domain leads to these operations to be implemented as in Table 1. In the case where other operations that are more complex are needed, many different implementations are found in the literature (division [24], square roots [24], reversible gates [25], *etc.*), though they are not to be presented here to focus on the implementation procedure of the studied system, which only needs additions (substraction) and multiplications.

Related to the conversion between BEN and SEN, it is usually implemented using a N-bit random number generator (RNG), whose output is compared to the value of the N-bit BEN. If the RNG number is below the BEN, the binary output is 1, or 0 otherwise, as described in Table 1. The complementary operation, SEN converted back to its BEN representation, a counter is used to determine the number of 1 in the string, which is a estimation of the probability. This obviously presents some error, as discussed below.

2.2. Error Estimation

Stochastic numbers are equivalent to bimodal processes. The emerging error when considering the approximation of a SEN to the value it represents can be calculated using a random walk process of length n. Thus, it is proportional to $\sqrt{n}$ [26]. This way, in an N bit binary number, the noise caused by the random walk process can be considered to be included in the lowest $N/2$ bits, and the relation between the power S_p of the signal and the power N_p of the noise (i.e., the noise figure NF) will be:

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

This NF is a key parameter to determine the required number of bits. It is also closely related to the sensitivity of the equations system to noise. Its has been shown that linear equations can still behave correctly using a low N, but nonlinear systems need higher values in order to reproduce a correct behavior [27]. On the other hand, it is also worth noting that this representation error also be interpreted as a random noise of amplitude $2^{1-N/2}$, assuming that we are using the $[-1,1]$ range. For instance, a 16-bit implementation would lead to an equivalent noise amplitude of $2^{-7} \approx 0.008$ in the mentioned range.

2.3. Implementation of Basic Differential Equations

The process of implementing differential equations using SC requires rewriting them in a specific way [27]. Specifically, the three different transformations below are needed:

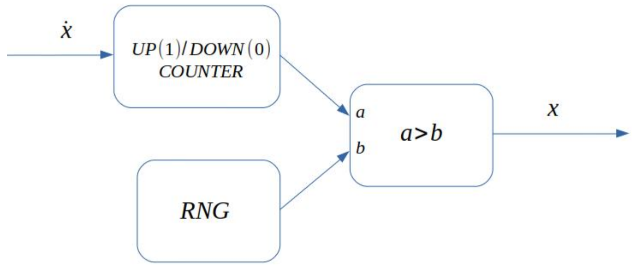

Figure 1. Basic implementation scheme of a SC integrator. Notice that both the input $\dot{x}(t)$ and the output $x(t)$ are SEN numbers.

1. The equation terms must be organised in a form suited to SC. As an example, the additions must be replaced by half additions: $a + b \rightarrow (2a + 2b)/2$, while multiplications remain the same. In the case where more complex operations are needed, an expansive reworking of the equations may be needed to ensure all the operations can be implemented in SC in the $[-1, 1]$ or [0,1] range.
2. All the variables have to fall inside the chosen domain ($[-1..1]$ or [0..1]).
3. Any remaining coefficient must be below 1 (in absolute value). This is achieved by using a time scaling.

After applying steps above, the equations may be processed as a SC system, i.e., multiplications and additions are to be implemented according Table 1.

Another of the elements needed to implement ODE is an integrator. It can be realized easily using the description in Figure 1. The circuit there performs a continuous integration, which is implemented by using a counter that increases or decreases one unit depending on whether the input is 1 or 0. The value of this counter is a BEN, which is then converted to SEN.

Time step Δt in the simulation is related to the number of bits, and this is dependent on which integration method is used. Using a single first order integrator, the effective time step is determined as [27]:

$$\Delta t = \frac{N_{acc}}{2^N}. \tag{2}$$

Notice that the design of the integrator includes deciding on the number of bits it has, this being equivalent to set the precision of the integrator, with a noise figure provided by Equation (1). In relation with the number of RNG that are used for this scheme, [28] shows that the use of the same RNG in both inputs leads to an actual improvement of the accuracy.

3. The Hindmarsh–Rose Model

3.1. Original Model

The Hindmarsh-Rose model [29,30] was proposed in the first half of the 1980's to describe the membrane potential $x(t)$ of a neuron, coupled to the rate of transport of sodium and potassium ions $y(t)$, as well as to a so-called adaptation current $z(t)$. The equations proposed by this model are as follows, when expressed in the dimensionless form:

$$\frac{dx}{dt} = y - ax^3 + bx^2 - z + I \tag{3}$$

$$\frac{dy}{dt} = c - dx^2 - y \tag{4}$$

$$\frac{dz}{dt} = r(s(x - x_R) - z) \tag{5}$$

```
f10 = mult(get_sn(x1),Y0)), neg(mult(get_sn(x2),X0));
f11 = add(add(get_sn(x0), f10), f10);
f12 = add(mult(get_sn(x3),X1), neg(mult(get_sn(x4),X2)));
f13 = add(neg(mult(get_sn(x5),Z0)), get_sn(u) );
f14 = add(f12,f13);
x = add(f11,f14);
f20 = neg(add(get_sn(y0), mult(get_sn(y1),X3)));
f21 = add(mult(get_sn(y2),X4), neg(mult(get_sn(y3),Y1)));
y = add(f20,f21);
f30 = add(mult(get_sn(z1),X5), neg(mult(get_sn(z2),Z1)));
z = add(neg(get_sn(z0)), f30);
```

Figure 2. Example of code used to implement Eq. 3-5 in a form suited to Stochastic Computing. The auxiliary functions are specified in Table 1. The values (lowercase) x0..x5, y0..y3, z0..z2 are the values of the parameters in the transformed equations.

The usual values of the dimensionless parameters are a=1, b=3, c=1, d=5, r=10^{-3}, s=4, x_R=-1.6. The first four (a,b,c,d) are used to model the behaviour of the fast ion channels, while r models the slow ion channels. The time scale of the neuron is defined by r, and x_R is related to the membrane potential. The value of the forcing current into the neuron I is around -10 to 10, and we have used I=3. Some procedures to fit these parameters to actual measured neuronal behaviour can be found in the literature [31–33].

3.2. Stochastic Computing Implementation

In order to implement the model, equations 3-5 must be rewritten in a suitable form, as discussed above. That is, we have to ensure the following conditions [27]:

1. All the additions are expressed in a suitable form for stochastic computing.
2. All the values of x,y and z are inside the [-1,1] interval.
3. All the values of the parameters are inside the [-1,1] interval.

The code we have used to implement the HR model after those transformations is shown in Fig. 2. The auxiliary functions add(), mult(), neg() and get_sn() are defined in Table 1. Notice that, to eliminate the problems associated to correlation when using a squared variable, we use different stochastic values each time we need to use x (X0, X1, X2, X3, X4, X5), y (Y0, Y1), and z (Z0, Z1). All these values obviously represent the same values but, due to the probabilistic nature of the stochastic representation, are different chains of 0 and 1.

4. Results

All the results in this section have been obtained using Matlab simulation. The corresponding code is freely available at https://github.com/rpicos-uib/stochastic_nonlinear_chaos, in the HR_model folder.

The first batch of results concern the number of bits (i.e. the length of the chain) needed to obtain the expected behaviour. That is, we expect to be able to reproduce spiking. To do so, we first integrated the HR model using conventional arithmetic, for the set of parameters a=1, b=3, c=1, d=5, r=10^{-3}, s=4, x_R=-1.6. For the sake of convenience, we will call this simulation the *exact* solution. We integrated up to a final time of 100s with dt=0.01s, using the transformed equations discussed in the previous section, and initial conditions x=0.351, y=0.862, z=0.666, which are equivalent to the non-scaled x=0.1, y=0.1, z=3. Notice that, since the scaling is different for the different variables, the initial conditions do not scale equally. The results for the x variable are shown in Fig. 3. We can see that the spiking behaviour is present, with amplitudes close to 0.6 and increasing distance between them. The phase space representation of this system (the x-y variables) is shown in Fig. 4.

After the exact simulation, we have tested three different lengths l_B of the SEN. Specifically, we have tested for N_b=20, 19, 16, where $l_B = 2^{N_b}$. The temporal evolution of

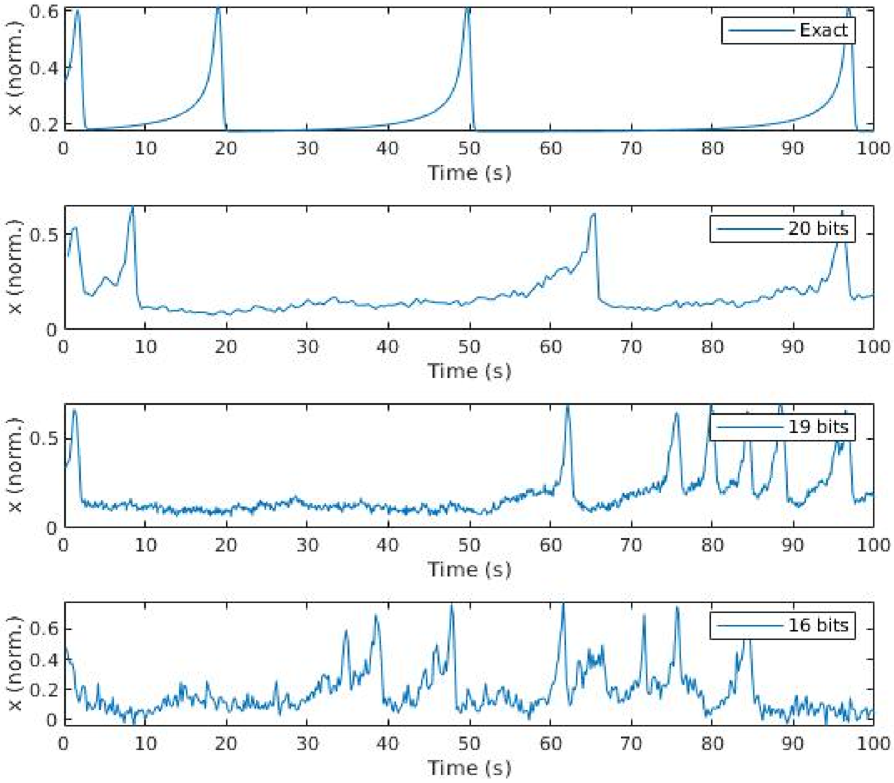

Figure 3. Temporal evolution of the x variable in the HR model, using conventional arithmetic (top, "exact"), and stochastic computing (20, 19, 16 bits).

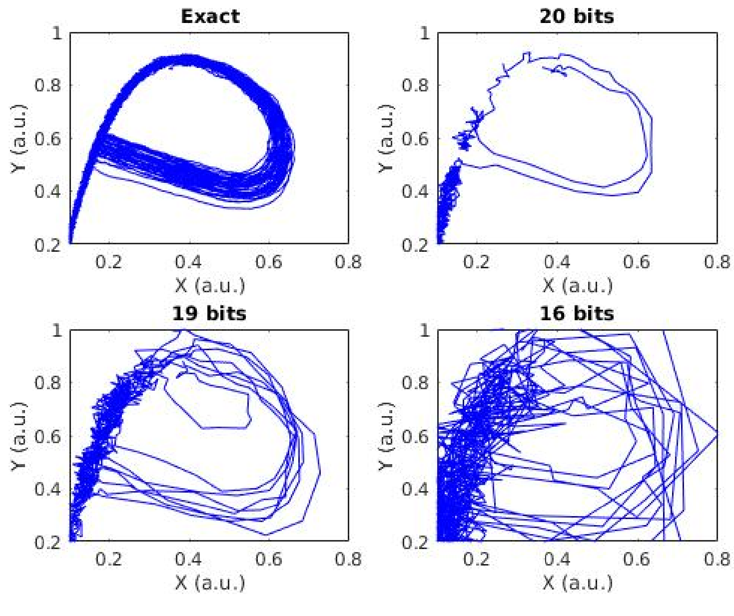

Figure 4. Space state of the x and y variables in the HR model, using conventional arithmetic (top, "exact"), and stochastic computing (20, 19, 16 bits).

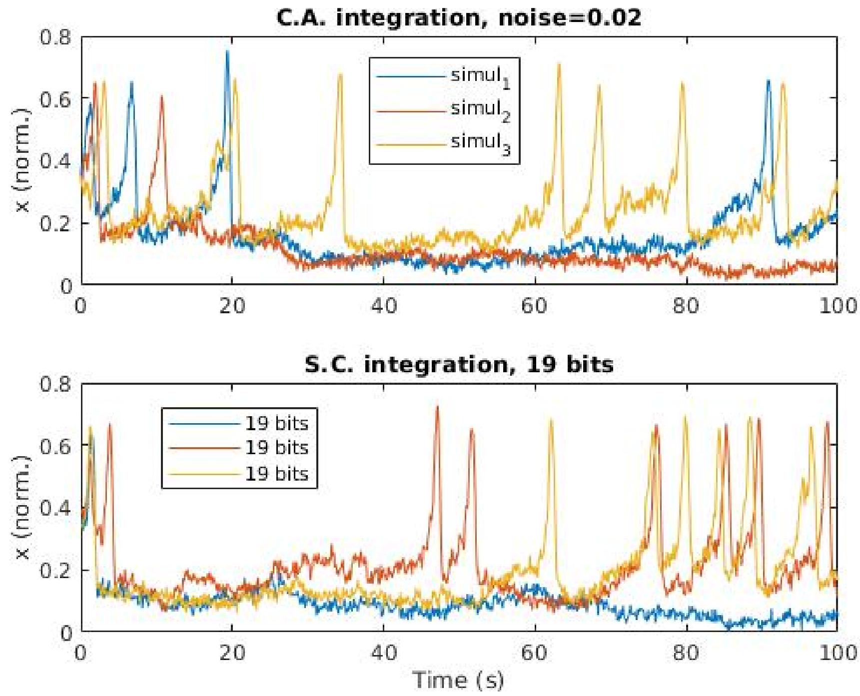

Figure 5. Comparison between three different simulation using the exact (Conventional Arithmetics, C.A.) solution (top) with $a_\epsilon = 0.02$ noise, and three different simulations using SC and 19 bits (bottom).

these solutions are depicted in Fig. 3, and the space states are plotted in Fig. 4. All of these simulations were obtained using the same initial conditions and parameter values than for the exact solution.

As is clear from the Figures, the spiking behaviour is present in all three SC simulations. The results corresponding to 16 bits seem too much noisy, and the space state portrait is hard to distinguish. However, at 19 and 20 bits, the signal is much clearer, and the portrait is obviously present. In addition, the temporal evolution of the signal is less noisy.

It is worth noticing that the temporal evolution of the signals is quite different from the exact solution. This could be expected, since one of the main characteristics of nonlinear systems is that small differences at the input cause enormous differences in the dynamical evolution. As discussed above, we have noise in all of the parameters of the system: the variables and the constants. That is, we can assume that we have a noise signal overlapping the integrator, as mentioned when discussing the noise in SC systems. We have checked the effect of noise in the exact solution by introducing three noise sources (ϵ_x, ϵ_y, ϵ_z) in the HR equations:

$$\frac{dx}{dt} = y - ax^3 + bx^2 - z + I + \epsilon_x \tag{6}$$

$$\frac{dy}{dt} = c - dx^2 - y + \epsilon_y \tag{7}$$

$$\frac{dz}{dt} = r(s(x - x_R) - z) + \epsilon_z \tag{8}$$

The noise sources are white noise generators of constant amplitude a_ϵ. Figure 5 compares three different simulation using the exact solution (top) with $a_\epsilon = 0.02$ noise, and three different simulations using SC and 19 bits (bottom). As can be seen in the exact solution, even such a small noise affects enormously the dynamics.

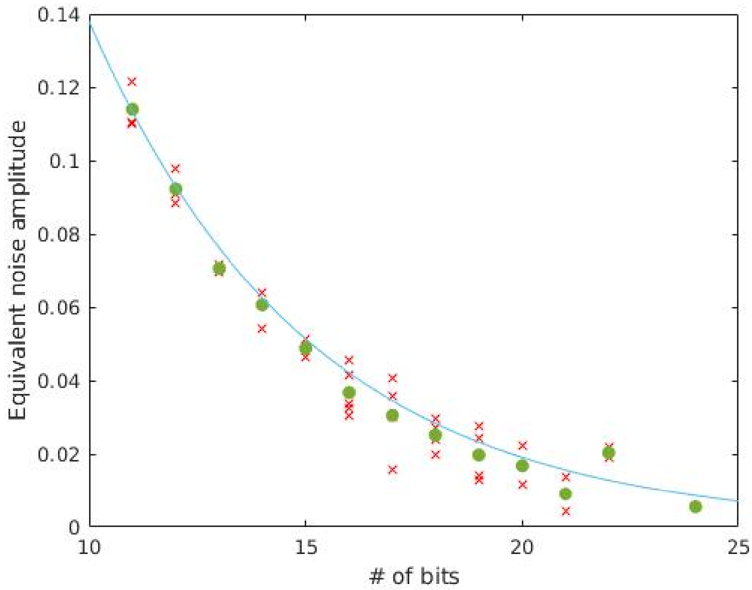

Figure 6. Equivalent noise amplitude for different number of bits. The red x correspond to the noise level of a single simulation; the green o are the average of all the simulations with a given number of bits; the blue line is the fitting of the averages according to Eq. 9, with $\eta = 1/3.5$.

To further study this equivalence, we have performed several simulations for different numbers of bits, between 11 and 24. The equivalent noise level has been calculated as the rms noise when the x variable is not spiking, since in the exact solution this would correspond to a zero value. The results are shown in Fig. 6, where the noise equivalent level for each simulation is shown as a cross, and an average for all the simulations corresponding to a given number of bits are shown as a green circle. It can be seen that the tendency is downwards, as expected, with a noise following a power law as in Eq. 9:

$$\epsilon_x = 2^{-\eta N}. \tag{9}$$

where η is a parameter that accounts for the effects on nonlinearity and feedback on the system. In our case, η has been found to be $\eta = 1/3.5$.

5. Discussion

In this paper we have presented a discussion on the implementation of the HR neuron model using Stochastic Computing. This approach has benefits considering the ease of implementation, requiring a very low number of gates, since arithmetic operations can be carried out using simple logic operations, as can be XNOR for multiplication or a multiplexer for addition. In order to implement the equations, we have rewritten them, as stated in [27], to be normalized in the range [-1,1] for all the possible mathematical operations, including values of the constants and the time scale.

These modified equations have been implemented in Matlab and simulated for different number of SCN lengths. It has been found that SC implementation is equivalent to introducing a noise source in the original equations. We have empirically found that the relation between the amplitude of the introduced noise and the length of the SCN follows an inverse power low as this is expressed in Eq. 9. We have compared this intrinsic noise to the effect of introducing a noise in the conventional arithmetic equations, and we have seen that, for instance, at a 0.02 noise level (normalized to $[-1,1]$), we have very similar results to those in SC using 18 bits.

Related to the speed of a possible ASIC or FPGA realization, we can state that we need 2^N clock cycles to perform a full operation equivalent to a conventional arithmetic implementation. That is, we can do the following number of steps per unit of time:

$$N_O = \frac{2^N}{T_{clk}}. \tag{10}$$

In our case, we have seen that using 19 bits provided a good enough approximation. Thus, assuming a FPGA running at 100MHz and with the same $dt = 0.01$s as used in the simulations, we would need t_O real time seconds to get one second in simulation time:

$$t_O = 2^N T_{clk} \frac{1s}{dt}. \tag{11}$$

This results in $t_O \approx 0.52$s. This way, a very simple implementation can provide a x2 speedup with reference to a real time system, including noise. This would pave the way to *in silico* simulation of simple biological systems, since the low number of required gates per neuron allows for complex systems. However, it has to be noted that the speed of the simulated systems would be closer to the biological originals, which take advantage of parallelism instead of just speed.

In any case, it has to be noted that simulation of an equivalent system of a single neuron using Matlab is much faster, requiring only on the order of ms. This must be, however, taken *cum grano salis*, since the required time in Matlab for a large number of neurons grows very fast, while in SC using an actual implementation in a FPGA would not. This is caused because the implementation in SC runs in parallel, thus the required time is not growing with the number of neurons, even if the required number of components does.

As a comparison with an equivalent implementation in hardware, we can mention a FPGA implementation of a simpler version of the HR model in 2 dimensions, using piece-wise linear approximations of the functions [34]. This work used a 32 bits fixed point arithmetic implementation requiring 4 full adders and 2 multiplexers for their preferred option. Another implementation in an FPGA was performed in [35], which implemented a fractional-order version of the HR model into an Altera DE2-115 FPGA. Their implementation for a single neuron required 1425 logic elements, and 1589 registers. As a comparison, our implementation just required 70 logic gates for the 14 adders, 20 XNOR gates for the multiplications and negations, plus the random number generator, which can be done externally using a white noise generator. It has to be noted that this reduction in the number of needed elements has the drawback of a much higher run time, as discussed above.

Future work will follow this line, implementing whole layers of neurons to study, among others, epileptic seizures [36]. Currently, most of the simulations are performed using numerical simulations that require very long times, so our approach would be highly beneficial in this field, allowing real time simulation.

Author Contributions: All the authors participated in all stages of this paper. All authors have read and agreed to the published version of the manuscript.

Funding: Part of this work was funded under the Spanish Ministerio de Economía y Competitividad DPI2017-86610-P and TEC2017-84877-R (also supported by the FEDER program).

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Not applicable

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
B2S	Binary to Stochastic
BEN	Binary Encoded Number
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IoT	Internet of Things
NF	Noise Figure
ODE	Ordinary Differential Equation
RNG	Random Number Generator
SC	Stochastic Computing
SCN	Stochastic Computing Number (see also SEN)
SEN	Stochastic Encoded Number(s)

References

1. Cattell, R.; Parker, A. Challenges for brain emulation: why is building a brain so difficult. *Natural intelligence* **2012**, *1*, 17–31.
2. D'Angelo, E.; Jirsa, V. The quest for multiscale brain modeling. *Trends in Neurosciences* **2022**.
3. Blue, C. Could Quantum Computing Revolutionize Our Study of Human Cognition? *APS Observer* **2021**, *34*.
4. Camuñas-Mesa, L.A.; Linares-Barranco, B.; Serrano-Gotarredona, T. Neuromorphic spiking neural networks and their memristor-CMOS hardware implementations. *Materials* **2019**, *12*, 2745.
5. Kaiser, J.; Billaudelle, S.; Muller, E.; Tetzlaff, C.; Schemmel, J.; Schmitt, S. Emulating Dendritic Computing Paradigms on Analog Neuromorphic Hardware. *Neuroscience* **2021**, *489*. doi: 10.1016/j.neuroscience.2021.08.013.
6. Höppner, S.; Mayr, C. Spinnaker2-towards extremely efficient digital neuromorphics and multi-scale brain emulation. *Proc. NICE* **2018**.
7. Schemmel, J.; Kriener, L.; Müller, P.; Meier, K. An accelerated analog neuromorphic hardware system emulating NMDA-and calcium-based non-linear dendrites. 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017, pp. 2217–2226.
8. Yamaura, H.; Igarashi, J.; Yamazaki, T. Simulation of a Human-Scale Cerebellar Network Model on the K Computer. *Frontiers in Neuroinformatics* **2020**, *14*. doi:10.3389/fninf.2020.00016.
9. Ellingsrud, A.J.; Boullé, N.; Farrell, P.E.; Rognes, M.E. Accurate numerical simulation of electrodiffusion and water movement in brain tissue. *Mathematical Medicine and Biology: A Journal of the IMA* **2021**, *38*, 516–551.
10. Schirner, M.; Kong, X.; Yeo, B.T.; Deco, G.; Ritter, P. Dynamic primitives of brain network interaction. *NeuroImage* **2022**, *250*, 118928. doi:https://doi.org/10.1016/j.neuroimage.2022.118928.
11. Bunruangses, M.; Youplao, P.; Amiri, I.S.; Pornsuwancharoen, N.; Yupapin, P. Brain sensor and communication model using plasmonic microring antenna network. *Optical and Quantum Electronics* **2019**, *51*, 1–10.
12. Amunts, K.; Ebell, C.; Muller, J.; Telefont, M.; Knoll, A.; Lippert, T. The human brain project: creating a European research infrastructure to decode the human brain. *Neuron* **2016**, *92*, 574–581.
13. Schirner, M.; Domide, L.; Perdikis, D.; Triebkorn, P.; Stefanovski, L.; Pai, R.; Popa, P.; Valean, B.; Palmer, J.; Langford, C.; Blickensdörfer, A.; van der Vlag, M.; Diaz-Pier, S.; Peyser, A.; Klijn, W.; Pleiter, D.; Nahm, A.; Schmid, O.; Woodman, M.; Zehl, L.; Fousek, J.; Petkoski, S.; Kusch, L.; Hashemi, M.; Marinazzo, D.; Mangin, J.F.; Flöel, A.; Akintoye, S.; Stahl, B.C.; Cepic, M.; Johnson, E.; Deco, G.; McIntosh, A.R.; Hilgetag, C.C.; Morgan, M.; Schuller, B.; Upton, A.; McMurtrie, C.; Dickscheid, T.; Bjaalie, J.G.; Amunts, K.; Mersmann, J.; Jirsa, V.; Ritter, P. Brain Modelling as a Service: The Virtual Brain on EBRAINS, 2021. doi:10.48550/ARXIV.2102.05888.
14. Gleeson, P.; Cantarelli, M.; Marin, B.; Quintana, A.; Earnshaw, M.; Sadeh, S.; Piasini, E.; Birgiolas, J.; Cannon, R.C.; Cayco-Gajic, N.A.; Crook, S.; Davison, A.P.; Dura-Bernal, S.; Ecker, A.; Hines, M.L.; Idili, G.; Lanore, F.; Larson, S.D.; Lytton, W.W.; Majumdar, A.; McDougal, R.A.; Sivagnanam, S.; Solinas, S.; Stanislovas, R.; van Albada, S.J.; van Geit, W.; Silver, R.A. Open Source Brain: A Collaborative Resource for Visualizing, Analyzing, Simulating, and Developing Standardized Models of Neurons and Circuits. *Neuron* **2019**, *103*, 395–411.e5. doi: https://doi.org/10.1016/j.neuron.2019.05.019.

15. Sy, M.F.; Roman, B.; Kerrien, S.; Mendez, D.M.; Genet, H.; Wajerowicz, W.; Dupont, M.; Lavriushev, I.; Machon, J.; Pirman, K.; others. Blue Brain Nexus: An open, secure, scalable system for knowledge graph management and data-driven science. *Semantic Web*, pp. 1–31.
16. Gao, M.; Wang, Q.; Arafin, M.T.; Lyu, Y.; Qu, G. Approximate computing for low power and security in the internet of things. *Computer* **2017**, *50*, 27–34.
17. Von Neumann, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies* **1956**, *34*, 43–98.
18. 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.
19. 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.
20. Ardakani, A.; Leduc-Primeau, F.; Onizawa, N.; Hanyu, T.; Gross, W.J. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2017**, *25*, 2688–2699.
21. Gaines, B.R. Stochastic computing systems. *Advances in information systems science* **1969**, pp. 37–172.
22. Gross, W.J.; Gaudet, V.C. *Stochastic Computing: Techniques and Applications*; Springer, 2019.
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. 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.
25. 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/p doi:10.1002/jnm.2711.
26. 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.
27. Camps, O.; Stavrinides, S.G.; Picos, R. Stochastic computing implementation of chaotic systems. *Mathematics* **2021**, *9*, 375.
28. Liu, S.; Gross, W.J.; Han, J. Introduction to Dynamic Stochastic Computing. *IEEE Circuits and Systems Magazine* **2020**, *20*, 19–33.
29. Hindmarsh, J.; Rose, R. A model of the nerve impulse using two first-order differential equations. *Nature* **1982**, *296*, 162–164.
30. Hindmarsh, J.L.; Rose, R. A model of neuronal bursting using three coupled first order differential equations. *Proceedings of the Royal society of London. Series B. Biological sciences* **1984**, *221*, 87–102.
31. Mukae, J.; Totoki, Y.; Suemitsu, H.; Matsuo, T. Parameter and input estimation in Hindmarsh-Rose neuron by adaptive observer. 2011 IEEE/SICE International Symposium on System Integration (SII), 2011, pp. 1090–1095. doi:10.1109/SII.2011.6147601.
32. Fradkov, A.L.; Kovalchukov, A.; Andrievsky, B. Parameter Estimation for Hindmarsh–Rose Neurons. *Electronics* **2022**, *11*, 885.
33. Beyhan, S. Affine TS fuzzy model-based estimation and control of Hindmarsh–Rose neuronal model. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2017**, *47*, 2342–2350.
34. Heidarpur, M.; Ahmadi, A.; Kandalaft, N. A digital implementation of 2D Hindmarsh–Rose neuron. *Nonlinear Dynamics* **2017**, *89*, 2259–2272.
35. Malik, S.; Mir, A.H. Discrete multiplierless implementation of fractional order hindmarsh–rose model. *IEEE Transactions on Emerging Topics in Computational Intelligence* **2020**, *5*, 792–802.
36. Peng, Y.; Jian, Z.; Wang, J. Study on discharge patterns of Hindmarsh-Rose neurons under slow wave current stimulation. International Conference on Natural Computation. Springer, 2006, pp. 127–134.