Analog Hardware Implementation of the Random Neural Network Model

Hossam Abdelbaki, Erol Gelenbe
School of Computer Science
University of Central Florida
Orlando, FL 32816
{ahossam, erol}@cs.ucf.edu

Said E. El-Khamy
Department of Electrical Engineering
Alexandria University
Alexandria, Egypt, 21544
elkhamy@alex.eun.eg

Abstract

This paper presents a simple continuous analog hardware realization of the Random Neural Network (RNN) model. The proposed circuit uses the general principles resulting from the understanding of the basic properties of the firing neuron. The circuit for the neuron model consists only of operational amplifiers, transistors, and resistors, which makes it candidate for VLSI implementation of random neural networks with feedforward or recurrent structures. Although the literature is rich with various methods for implementing the different neural networks structures, the proposed implementation is very simple and can be built using discrete integrated circuits for problems that need a small number of neurons. A software package, RNNSIM, has been developed to train the RNN model and supply the network parameters which can be mapped to the hardware structure. As an assessment on the proposed circuit, a simple neural network mapping function has been designed and simulated using PSpice.

1 Introduction

One interesting feature of neural networks is that complex tasks like pattern recognition can be performed more rapidly by animals than by large digital computers even though the processes in digital computers are much faster than the ionic processes occurring in neural networks. There are three main approaches to perform real-life neural computations. The first one is the software simulation using sequential computers, the second approach is the parallel computation using multiprocessor system, and the third is the realization of the network in hardware using special purpose digital or analog components. The first alternative at present is the most easily accessible because of the availability of personal computers and developing software tools. Since software programs are designed to run on single processor computer, this approach contradicts the basic principle of neural network, which lies in its massive parallelism. In the second approach, a neural network can be emulated by a very large number of simple processing elements, popular examples are networks of transputers. However, most of the current work involving parallel processing maintains synchrony by means of a system clock while synchrony is absent from neuron spike trains in various neural networks. The absence of clocking in neural networks reflects the fact that coding of information is different from information coding in digital computers. Finally, the designer has to decide between analog and digital implementation.

The random neural network [1, 2] is a biologically inspired spiked model which differs substantially from existing deterministic models such as the MLP. The network has a compact closed form solution for network state even in the recurrent case, this in turn yields efficient numerical algorithms. Typically, a spiked stochastic model will include some internal representation of each neuron's state and a probabilistic representation of successive firing times as a function of state; additionally, rules need to be given about the manner in which the internal state changes after firing. In this model, the internal state is a nonnegative integer; it rises or falls depending on the excitatory or inhibitory nature of incoming spikes, and it drops each time the neuron fires. Interfiring are exponentially distributed. This is a recurrent network model which may have feedback loops of arbitrary topology. The general training algorithm was presented in [3]. By appropriately mapping external signals and neuron states into certain physical quantities, the RNN has been successfully applied to several engineering problems [5, 6, 8]

0-7695-0619-4/00 $10.00 © 2000 IEEE

In [4], a digital neuron realization for the RNN has been proposed using discrete logic integrated circuits. Although that implementation represents clearly the firing mechanism in the RNN neuron, it is so complex to be built and maintain the synchronization between the different parts of the design. Also, it is dealing with only digital signals. As it turns out, neural networks by their nature tend to favor the analog approach. In this paper, a simple analog implementation of the RNN neuron is proposed. This approach needs only the addition and multiplication operations and for small networks, it can be built with off-the-shelf components. It should be noted here that both the digital implementation and the proposed analog implementation suppose that the network has been trained off line using a software program [9] and after convergence, the parameters resulted from the training process are fed to the model.

The Sections of this paper are organized as follows. Section 2 presents the proposed neuron analog implementation. In Section 3, a PSpice simulation is carried out to illustrate how the neuron model can be used in a simple mapping network. Finally, Section 4 gives the conclusions.

2 Analog Modeling of the RNN neuron

Recalling the basic equations of the RNN model from [1, 3] we have:

$$q_i = \frac{\lambda_i^+}{r_i + \lambda_i^-} \tag{1}$$

where the λ_i^+ and λ_i^-, for $i = 1, \ldots, n$ satisfy the system of non-linear simultaneous equations:

$$\lambda_i^+ = \Lambda_i + \sum_{j=1}^{n} q_j w_{ji}^+, \qquad \lambda_i^- = \lambda_i + \sum_{j=1}^{n} q_j w_{ji}^-, \tag{2}$$

$$r_i = \sum_{j=1}^{n} w_{ij}^+ + w_{ij}^- \tag{3}$$

where q_i is the output of neuron i and w^+ and w^- are the excitatory and inhibitory weights respectively. It is obvious from the equations that modeling the RNN neuron requires only the addition, multiplication, and division (subtraction is not needed since all the external and internal quantities involved with the model are positive floating point numbers). Assuming that the network training has been completed successfully using a computer program, the resulting inhibitory and excitatory weight matrices can be used to design and implement the network under consideration. For illustrating the proposed neuron model, a recurrent network consisting of three neurons will be considered. Figure 1 shows the internal structure of neuron 1 which can be connected in general to the other two neurons with feedback weights.

The operational amplifier U1C accepts the external inhibitory input λ_1 (lambdas1 in the Figure), the outputs of the other neurons q_1 and q_2, and the firing rate r_1. q_1 is connected via the inhibitory weight w_{21}^- (wm21 in the Figure), similarly, q_3 is connected via the weight w_{31}^- (wm31 in the Figure). The rate r_1 is calculated from the weights using Equation 3 and its value is represented by the fixed voltage given by $V1$. The inverted sum output appears at node 8 of U1C. U1D acts as an unity gain inverting buffer so that it produces λ_1^- at pin 7 as given in Equation 2.

In a similar manner, U1C and U1D act as a non inverting summing circuit for the external excitatory input Λ_1 (LAMBDAC1 in the Figure) and the weighted outputs from the other two neurons. The output at pin 7 of $U1B$ represents λ_1^+ as given in Equation 2. It should be noted here that every weight is inversely proportional to its corresponding resistor. As an example, if $w_{21}^- = 0.5$ then, the resistor $wm21$ will be equal to $10/0.5 = 20K$ and if $w_{31}^+ = 0$, the resistor $wp31$ should be ideally an open circuit and in the design, we represent the open circuit with a relatively large resistance value such as 1 Meg Ohms. The resistors representing the weights and the voltages representing the rates are expressed as symbolic variables. When using the neuron model in a certain RNN, the weights and rates under consideration will be passed to the model. This modular design facilitate using the same neuron model in different RNN structures.

IC U2 acts with the matched transistors as a log - antilog multiplier/divider circuit which can be used to divide the output of $U1B$ by the output of $U1D$. The output at pin 14 of $U2D$ represents the output q_1 of the neuron as

given in Equation 1. Several commercial multipliers such as $MPY100$ can also be used as reliable multipliers and dividers which may help when building networks with small number of neurons.

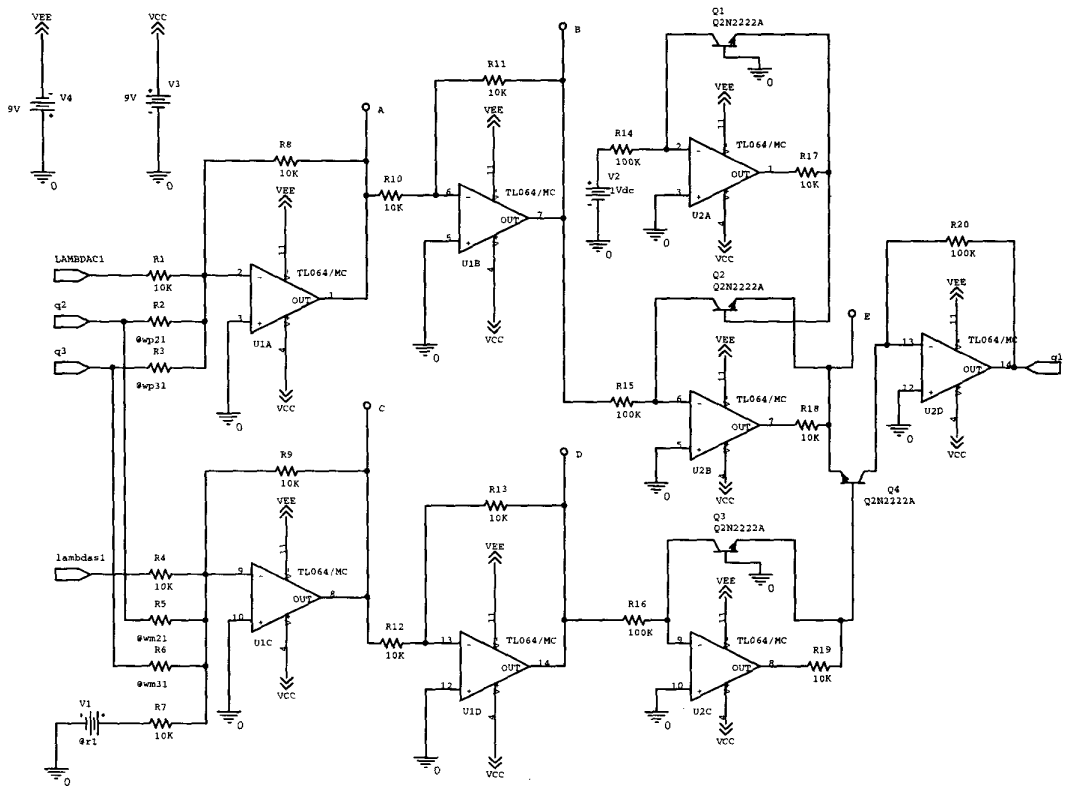

Figure 1: The proposed analog circuit for a single RNN neuron.

3 Simulation Results

To test the proposed neuron model, a mapping RNN with three neurons is simulated. This simple network accepts the signal x at its input and produces $\frac{1}{(1+x)^2}$, $0 \leq x \leq 1$, at its output. Although the function to be mapped is simple, it can be proved that the generalization of this network to calculate $\frac{1}{(1+x)^v}$ and $\frac{x}{(1+x)^v}$, $v > 0$, can be used to approximate any continuous multi variables function ([7], Theorem 5). From ([7] Lemma 2), we can calculate the network weights and rates as shown in Figure 2. Figure 3 illustrates the PSpice model of the mapping RNN.

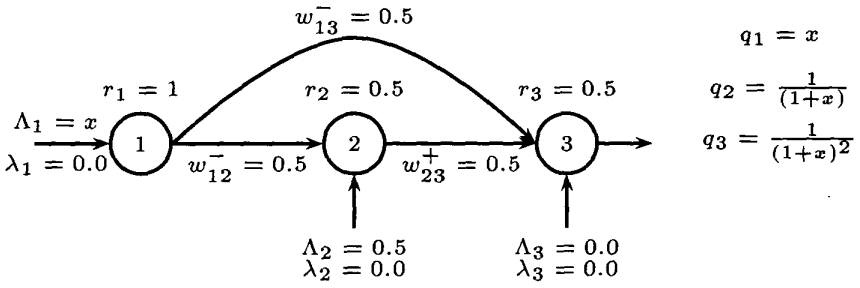

Figure 2: Simple mapping RNN.

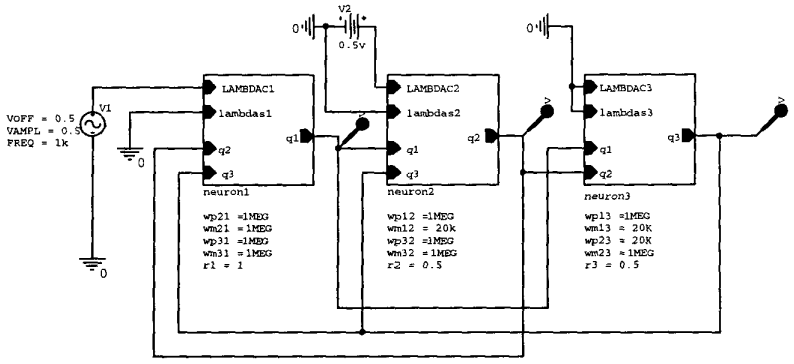

Figure 3: PSpice model for the mapping RNN.

Each of the three neuron modules represents the analog circuit shown in Figure 1 The parameters corresponding to each neuron (external inputs, rate, and weights to other neurons) are passed from the neuron module to the actual circuit. The network is driven by a 1KHz sine wave source with peak to peak value of 1V and offset of 0.5V. Although the network of Figure 2 is feed forward, the corresponding PSpice representation, as shown in Figure 3, is drawn as a recurrent network by taking care of the connection weights. For example, the output of neuron 3, q_3, is connected to the q_3 input of neuron 1 by the weights wp31 = wm31 = 1 Meg Ohms (or equivalently $w_{31}^{+} = w_{31}^{-} \simeq 0$). The output of each of the three neuron modules is shown in Figure 4.

In the given example the network parameters were calculated from the understanding of the network function but for large problems, training can be applied to obtain the values of the weights and rates [9], after that the model can be built and simulated using the individual neuron modules.

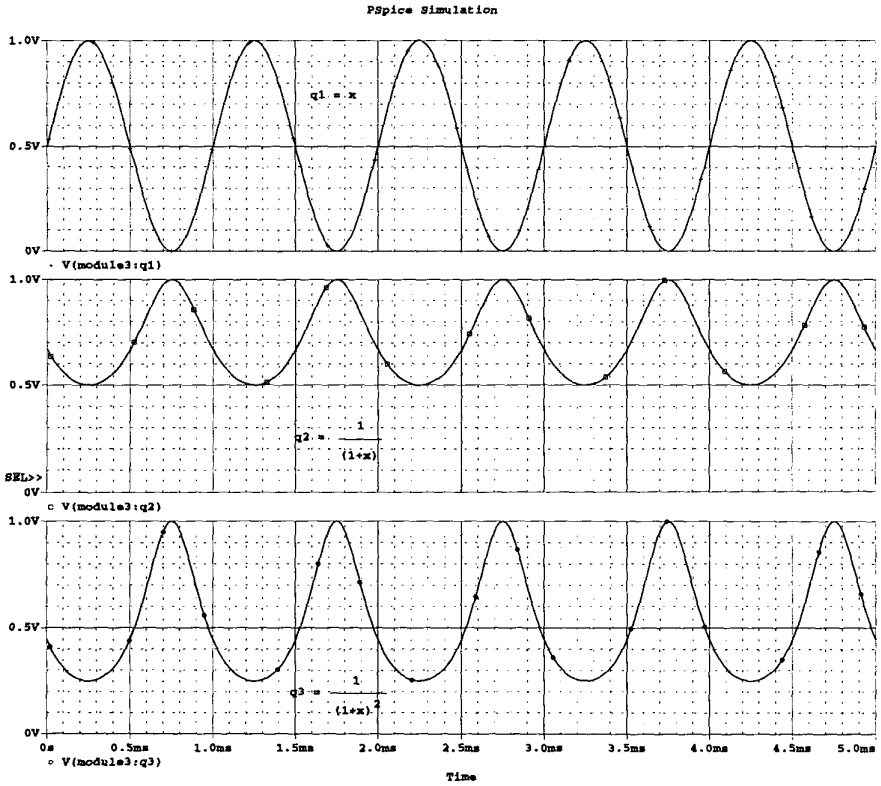

Figure 4: Simulation results for the mapping network.

4 Conclusion

In this paper, an analog circuit implementation for the random neural network is proposed and simulated using common operational amplifiers and transistors. It is worthy to point out that the implementation using these inexpensive discrete components, allows a convenient way to construct a prototype circuit to test the functionality of the neural network under consideration. In this work, verification of the implemented analog circuit has been proceeded using the PSpice circuit simulator. Given a specific application, the design using discrete components can be mapped directly to VLSI design. The discrete component design offers high flexibility, low productivity, large circuit area, high power consumption, and it can be used only for prototyping purposes. On the other hand, the VLSI approach offers low flexibility, high productivity, very small circuit area, low power consumption, and it can be used only for practical applications. This hardware realization brings the theoretical model into the practical applications. Although the proposed neuron design does not represent the transient response of the RNN neuron, it implements successfully its steady state behavior. One drawback of the hardware model is that it is incapable of on line training and weights update, but in the majority of practical problems, the training phase needs to be carried out off line using a dedicated software that runs on a computer [9] and after training, the final network parameters can be mapped directly into physical hardware components.

References

[1] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Computation,* vol. 1, no. 4, pp. 502-511, 1989.

[2] E. Gelenbe, "Stability of the random neural network model," *Neural Computation,* vol. 2, no. 2, pp. 239-247, 1990.

[3] E. Gelenbe, "Learning in the recurrent random neural network," *Neural Computation,* vol. 5, no. 1, pp. 154-164, 1993.

[4] C. Cerkez, I. Aybay, and U. Halici, "A digital neuron realization for the random neural network model," *Proceedings of the International Conference on Neural Networks,* Huston, pp.1000-1004, 1997.

[5] V. Atalay and E. Gelenbe, and N. Yalabik, "The random neural network model for texture generation," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 6, no. 1, pp. 131-141, 1992.

[6] C. Cramer, E. Gelenbe, and H. Bakircioglu, "Low bit rate video compression with neural networks and temporal sampling," *Proceedings of the IEEE,* vol. 84, no. 10, pp. 1529-1543, October 1996.

[7] E. Gelenbe, Z. H. Mao, and Y. D. Li, "Function approximation with spiked random networks," *IEEE Trans. on Neural Networks,* vol. 10, no. 1, pp. 3-9, 1999.

[8] H. Abdelbaki, E. Gelenbe, and S. El-Khamy, "Random neural network decoder for error correcting codes," *Proceedings of the International Joint Conference on Neural Networks,* Washington, DC, July, 1999.

[9] H. M. Abdelbaki, *Random Neural Network Simulator (RNNSIM) v.2,* Free simulator available at ftp://ftp.mathworks.com/pub/contrib/v5/nnet/rnnsimv2.