FPGA Implementation of Izhikevich Spiking Neural Networks for Character Recognition

Kenneth L. Rice[1] Mohammad A. Bhuiyan[1] Tarek M. Taha[2] Christopher N. Vutsinas[1] Melissa C. Smith[1]

[1]Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA
{krice, mbhuiya, cvutsin, smithmc}@clemson.edu
[1]Department of Electrical and Computer Engineering, University of Dayton, Dayton, OH 45469, USA
ttaha@ieee.org

Abstract— **There has been a strong push recently to examine biological scale simulations of neuromorphic algorithms to achieve stronger inference capabilities than current computing algorithms. The recent Izhikevich spiking neuron model is ideally suited for such large scale cortical simulations due to its efficiency and biological accuracy. In this paper we explore the feasibility of using FPGAs for large scale simulations of the Izhikevich model. We developed a modularized processing element to evaluate a large number of Izhikevich spiking neurons in a pipelined manner. This approach allows for easy scalability of the model to larger FPGAs. We utilized a character recognition algorithm based on the Izhikevich model for this study and scaled up the algorithm to use over 9000 neurons. The FPGA implementation of the algorithm on a Xilinx Virtex 4 provided a speedup of approximately 8.5 times an equivalent software implementation on a 2.2 GHz AMD Opteron core. Our results indicate that FPGAs are suitable for large scale cortical simulations utilizing the Izhikevich spiking neuron model.**

Keywords-FPGA;spiking neural networks

I. Introduction

The brain utilizes a large collection of slow neurons operating in parallel to achieve very powerful cognitive capabilities. There has been a strong interest amongst researchers to develop large parallel implementations of cortical models on the scale of animal or human brains. At this scale, the models have the potential to provide much stronger inference capabilities than current generation computing algorithms [1]. A large domain of applications would benefit from the stronger inference capabilities including speech recognition, computer vision, textual and image content recognition, robotic control, and data mining.

Several research groups are examining large scale implementations of neuron based models [2][3] and cortical column based models [4][5] on high performance computing clusters. IBM is utilizing a 32,768 processor Blue Gene/L system [2], while Los Alamos National Laboratory is utilizing the Roadrunner supercomputer (one of the fastest computers currently) to model the human visual cortex [6].

There is also a strong interest in the design of specialized hardware acceleration approaches for these neuromorphic algorithms to enable large scale simulations. The SpiNNaker project is developing an ARM based chip multiprocessor to evaluate 1000 leaky integrate-and-fire neurons [7]. Several researchers are examining the use of memristors [8][9] for the design of neural circuits [10]. Gao and Hammerstrom [11] proposed a simplified model of the neocortex based on spiking neurons and examined conceptual implementations of the model using future CMOS and CMOL technologies.

One of the most common set of algorithms being examined for large scale simulation and hardware acceleration is the spiking neural network (SNN) class of models. These models capture neuronal behavior more accurately than traditional neural networks. Several SNN models have been proposed recently. Of these, the integrate-and-fire model is the most commonly utilized – both for algorithmic studies and hardware implementations. FPGA implementations of the integrate-and-fire spiking neuron model include [12][13][14][15]. Shayai et al. [12] simulated a network of 161 quadratic integrate-and-fire neurons and 1610 synapses on a Virtex 5 FPGA. Upegui et al. [13] utilized a Spartan II FPGA to implement a network of neurons based upon a simplified integrate-and-fire neuron model. They also implemented Hebbian learning for the neurons in the FPGA. Cassidy et al. [14] developed a spiking network of leaky integrate-and-fire neurons to evaluate several experiments in a Spartan-3 FPGA. Pearson et al. [15] implemented a neural processor using leaky integrate-and-fire neurons to perform neural network computations on a Virtex II FPGA.

Izhikevich has shown [16] that the integrate-and-fire spiking neuron model is not very biologically accurate and is unable to reproduce the spiking behavior of many neurons. He proposed a new model [17] which has been shown to be almost as accurate as the highly detailed Hodgkin-Huxley neuron model but with the low computational cost of the integrate-and-fire model. Both the integrate-and-fire model and the Izhikevich model require 13 FLOPs per neuron simulation, while the Hodgkin-Huxley model requires 256 FLOPs. Therefore, for large scale simulations the Izhikevich model is significantly more attractive than the more commonly used integrate-and-fire.

Recent studies [18][19][20] have implemented the Izhikevich neuron model instead of the integrate-and-fire model on FPGAs. La Rosa et al. [18] and Fortuna et. al [19] simulated two such neurons on an FPGA. Their primary objective was to examine the feasibility of FPGA implementations of the model and to show that hardware can reproduce the wide range of neuronal responses possible from the model. Mokhtar et al. [20] simulated 48 neurons based on the Izhikevich model on a Virtex II Pro FPGA for maze navigation.

In this paper we explore the feasibility of using FPGAs for large scale simulations of the Izhikevich model. We utilized a character recognition algorithm based on the Izhikevich spiking neuron model presented in a previous paper [21]. The network in [21] was scaled up in this paper to evaluate the performance of large networks on FPGAs. The primary contributions of this work are:

1) The design of a modularized processing element (PE) for implementing the Izhikevich model on FPGAs. Multiple PEs can be placed on chip, with each PE being able to process a large number of neurons in a pipelined manner. The state of each neuron is stored in on-chip memory. In this study we implemented a network with over 9000 neurons using 25 PEs on a Xilinx Virtex 4 FPGA.

2) The use of the PEs to evaluate a specific application on FPGAs using the Izhikevich model. Our results indicate that the FPGA can provide a speedup of about 8.5 times over a software implementation on a 2.2 GHz AMD Opteron core.

Section II of this paper provides background on the Izhikevich model, while section III discusses the character recognition algorithm utilized. Section IV presents the hardware implementation of the Izhikevich character recognition algorithm. Sections V and VI present the experimental setup and results of the work, while section VII concludes the paper.

II. Background

Spiking neural models capture neuronal behavior more accurately than traditional neural models. A neuron consists of three functionally distinct components called dendrites, axons, and a soma. Each neuron is typically connected to over 10,000 other neurons. The dendrites of a neuron collect input signals from other neurons, while the axons send output signals to other neurons. Input signals coming in along dendrites can cause changes in the ionic levels within the soma, which can cause the neuron's membrane potential to change. If this membrane potential crosses a certain threshold, the neuron is said to have fired or spiked. In these events, the membrane potential rises rapidly for a short period of time (a spike) and causes electrical signals to be transmitted along the axons of the neuron to other neurons connected to it. Details of the spiking mechanism can be found in [22]. Spiking is the primary mechanism by which neurons communicate with each other. Over the last 50 years, several models have been proposed that capture the spiking mechanism within a neuron.

Izhikevich proposed a new spiking neuron model in 2003 [17] that is based on only two differential equations ((1) and (2)). In these equations, V is the membrane potential of the neuron, I is the synaptic current, and u represents a membrane recovery parameter used to supply negative feedback to the voltage.

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I \tag{1}$$

$$\frac{du}{dt} = a(bV - u) \tag{2}$$

$$\text{if } V \geq 30 \text{ mV, then } \begin{cases} V \leftarrow c \\ u \leftarrow u + d \end{cases}$$

By tweaking the four constant parameters (a, b, c, and d), the model can reproduce almost all types of neuronal responses seen in biological experiments. This makes the model almost as versatile as the Hodkin Huxley model at a fraction of the computational cost of that model. In this study we utilized the same timestep (1 ms) and model parameters used by Izhikevich in [16]. The values of the constant parameters are given in Appendix I, while Fig. 1 shows an example of spikes produced with this model.

III. Character Recognition Algorithm

In this study we utilized the two layer spiking neural network algorithm for character recognition based on the Izhikevich model presented earlier in [21]. In this model, the first layer acted as input neurons, and the second layer as output neurons. The network was trained as shown in [21] to recognize the 48 different input images shown in Fig. 2. These images represent the 26 upper case letters (A-Z), 10 numerals (0-9), 8 Greek letters, and 4 symbols.

Input images were presented to the first layer of neurons (which we refer to as level 1), with each image pixel corresponding to a separate input neuron. Thus the number of neurons in level 1 was equal to the number of pixels in the input image. The number of second layer neurons (which we refer to as level 2) was equal to the number of training images. This is because each level 2 neuron was encoded to fire only when it recognized one specific image. Lastly, each level 1 neuron was connected to every level 2 neuron. A small scale version of this network is shown in Fig. 3.

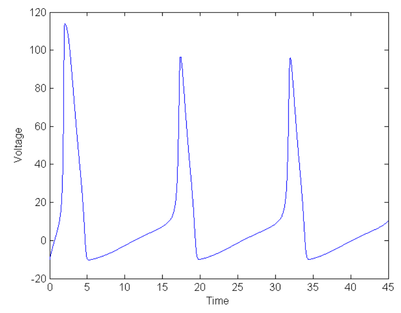

Figure 1. Spikes produced with Izhikevich model.

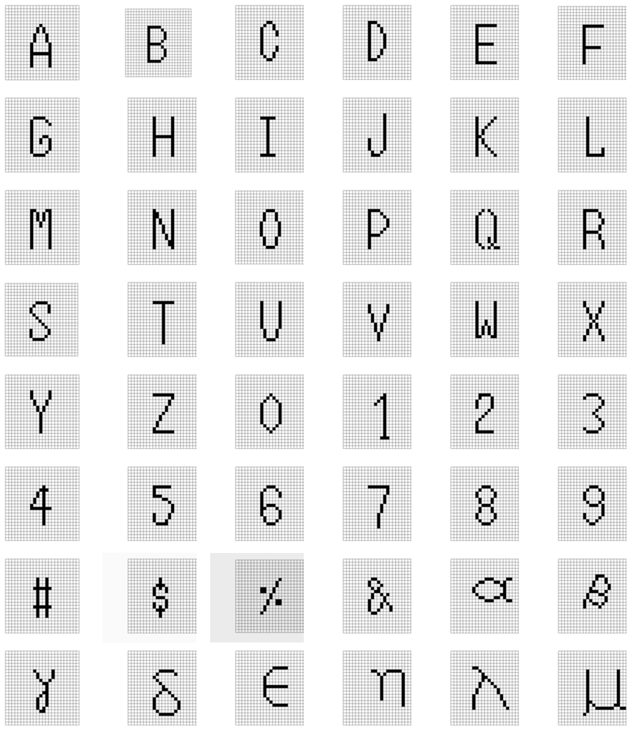

Figure 2. Training Images.

Each neuron has an input current *I* that is used to evaluate its membrane potential, *V*. If this membrane potential crosses a certain threshold (30 mV for our design) during a cycle, the neuron is considered to have fired. In case of a level 1 neuron, the input current, *I*, is zero if the neuron's corresponding pixel in the input image is "off". If the pixel is "on", a constant current is supplied to the input neuron. A level 2 neuron's overall input current is the sum of all the individual currents received from the level 1 neurons connected to it. This input current for a level 2 neuron is given by (3):

$$I_j = \sum_i w(i,j) f(i) \tag{3}$$

where,

w is a weight matrix in which *w(i,j)* is the input weight from the level 1 neuron *i* to the level 2 neuron *j*.

f is a firing vector in which *f(i)* is 0 if the i^{th} level 1 neuron does not fire, and is 1 is the neuron does fire.

The elements of the weight matrix *w* are determined through a training process where a set of training images are presented sequentially to the input neurons. The weight matrix thus obtained is used to determine the input current to each of the output neurons.

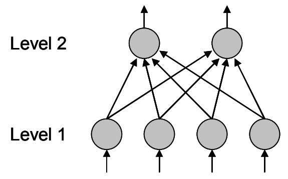

Figure 3. Illustration of network connections between level 1 and level 2 neurons.

Algorithm I. The recognition phase for the spiking neuron image recognition model

```
Repeat until a level 2 neuron fires:
    For all level 1 neurons:
        Read input current $I$
        Calculate neuron membrane potential $V$
        If neuron fires, append neuron index to firing vector
—Barrier—
    For all level 2 neurons:
        For each non zero entry of firing vector (previous cycle)
            Add corresponding weight element to input current $I$
            Calculate neuron membrane potential $V$
            If neuron fires, output is produced
—Barrier—
```

In the recognition phase, an input image is presented to the level 1 neurons and after a certain number of cycles (in this design 12 cycles is sufficient), one output neuron will fire, thus identifying the input image. During each cycle, the level 1 neurons are evaluated based on the input image and the firing vector is updated to indicate which of the level 1 neurons fired that cycle. In the same cycle, the firing vector generated in the previous cycle is used to calculate the input current *I* to each level 2 neuron. The level 2 neuron membrane potentials *V* are then calculated based on their input current *I*. This process is described in detail in Algorithm I. In this study, the Euler approach was utilized to solve equations (1) and (2).

IV. Hardware Implementation

A. Structure

The hardware implementation of the SNN character recognition algorithm was developed on a Cray XD1. Only the recognition aspect of the algorithm was accelerated in this study. An AMD Opteron processor and a Xilinx FPGA were utilized to perform the processing. Fig. 4 shows the overall design of this system. The FPGA is connected to the AMD processor through a high-speed interface logic module (adapted from designs provided by Cray) and to an external SRAM bank through a DMA interface module. The latter is used to transfer data between the off-chip SRAM banks and both the AMD Opteron and FPGA.

A third module on the FPGA, the SNN module, implements the character recognition algorithm. This consists of the following three components:

1) A set of SNN processing elements (PEs). These implement the neuron computations given by (1) and (2). Each PE evaluates a subset of the neurons and generates a local firing vector based on this subset of neurons.

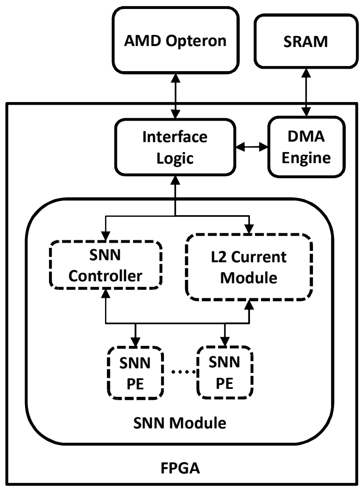

Figure 4. Overall Spiking Neural Network Design on Cray XD1.

2) An L2 current module. This implements the computations in (3) needed to generate the input currents for level 2 neurons. This involves examining all the local firing vectors of the level 1 PEs and then initiating the transfer of the level 2 neuron weights based on these firings.
3) An SNN controller. This unit coordinates the operations of the SNN PEs and the L2 current module to implement the character recognition algorithm.

B. *Processing Element*

Each SNN PE has a 23 stage pipeline and stores the parameters V, u, I, and f used in (1) and (2) locally in on-chip block RAM (BRAM). All SNN PEs are connected to the L2 current module through a bus, on which the L2 current module is the arbiter.

To reduce the logic resource footprint and to accelerate the operation of each PE, the computations in (1), (2), and (3) were implemented in fixed point format instead of floating point form. We found that fixed point representations with less than 12 bits after the radix point would produce incorrect character recognition because of high round off errors. Therefore a fixed point representation with 12 bits after the radix point was utilized in this study. The number of bits before the radix point increased from 4 to 32 bits as a result of the computations of the algorithm.

C. *Operation*

Before processing any images within the FPGA, the weight matrices *w(i,j)* for the level 2 neurons need to be initialized. The weights are pre-calculated during the training process and are stored initially in a file. These values are transferred into the SRAM associated with the FPGA through the DMA module (only once). The weights are stored in a 16 bit fixed point format with 12 bits after the radix point.

The interaction between the FPGA modules to evaluate the algorithm is coordinated by the SNN controller module. A state machine depicting the overall process of the SNN controller is shown in Fig. 5. The following section lists the interactions between the FPGA modules based on this figure.

1) *Initialization and startup:* The binary image to be recognized is transferred to the FPGA's on-chip BRAM through the interface logic module. Once the binary image data has been written, the AMD processor signals the SNN module on the FPGA to begin operation (step a in Fig. 5).

2) *Process neurons:* The PEs are designated to evaluate either level 1 or level 2 neurons. All the neurons for a given level are distributed evenly across the PEs for that level and are processed in this step (step b in Fig. 5). The objective of the processing is to determine a neuron's membrane potential, V. If this potential is 30 mV or higher, the neuron is considered to have fired. When a neuron fires, the PE stores the neuron's index in a local firing vector and also sets a "fired flag" on the PE.

3) *Examine level 2 firing vector:* In this step, each level 2 PE is examined to determine if its "fired flag" is set (step c in Fig. 5). The firing of a level 2 neuron indicates that a character was recognized. If the flag is set, processing ends (step g in Fig. 5), and the index of the level 2 neuron that fired is stored for later reading by the AMD processor. This index represents the character that was recognized.

4) *Examine level 1 firing vector:* In this step, each level 1 PE is examined in a round robin manner to determine if its "fired flag" is set (step d in Fig. 5). If any level 1 neuron fires, the computations in (3) have to be evaluated to determine the input current, I, for each level 2 neuron (step e in Fig. 5).

Weight computations involve streaming the indices of the level 1 neurons that fired (stored in the local firing vector of

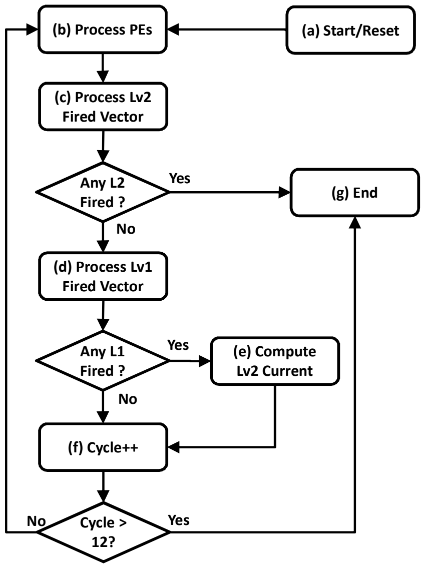

Figure 5. State machine for spiking neural network controller.

each PE) to the high speed off-chip SRAM. After an 11 cycle latency, this SRAM returns the level 2 neuron weights corresponding to the level 1 indices sent to it. A data read from the SRAM is a 64-bit word; therefore, four weight values (16 bits each), corresponding to four level 2 neurons, are packed together into each 64-bit SRAM word. The weight computations in (3) are also evaluated for four neurons at a time in the level 2 current module. The design was optimized to provide the SRAM with a continuous stream of indices to allow efficient processing of weights.

5) *Prepare for next simulation cycle:* The simulation cycle count is incremented (step f in Fig. 5). If this cycle count is less than 12, the SNN controller returns to step 2 to process another cycle. Otherwise, the SNN controller ends operations by setting a finished flag and reports the cycle number and image classification (index of level 2 node that fired) to the AMD processor.

V. Experimental Setup

We developed a fully software and a hardware-accelerated version of the SNN recognition algorithm on a Cray XD1 at the Naval Research Laboratory. The Cray XD1 consisted of 144 Xilinx Virtex II Pro FPGAs (part XC2VP50), 6 Xilinx Virtex 4 FPGAs (part XC4VLX160), and 864 AMD Opteron 2.2 GHz cores (432 dual core processors). The fully software implementation was written in C and was processed on a single AMD Opteron core. The code was compiled with *gcc* using the –O3 optimization.

Two character recognition networks were developed for this study. One was trained on the 48 binary 24×24 pixel images shown in Fig. 2. The second network utilized scaled versions of these images (scaled to 96×96 pixels). The structures of the two networks are shown in Table I. The networks were tested with their training images.

TABLE I. Sturcture of Neural Networks Examined

Model Parameters	**Networks Implemented**	
	1	***2***
Total SNN PEs	7	25
Input Image (pixels)	24×24	96×96
Level 1 neurons	576	9216
Level 2 neurons	48	48
Total neurons	624	9264

The two networks in Table I were implemented on two types of FPGAs on the Cray XD1. Network 1 was implemented on a Virtex II Pro FPGA, while network 2 on a Virtex 4 FPGA. On the Virtex II, we were able to accommodate six level 1 PEs and one level 2 PE. The Virtex 4, with approximately three times the amount of logic in a Virtex II Pro, was able to accommodate 25 level 1 PEs and one level 2 PE. The main difference between the level 1 and 2 PEs was the amount of block RAM (BRAM) dedicated to the firing vector and the values of the parameters a, b, c, and d in (1) and (2).

VI. Results

The resource utilization of the two FPGA implementations is shown in Table II. The systems were clocked at or near the 199 MHz frequency limit of the Cray XD1.

TABLE II. Device Logic Utilization

Utilization Metrics	**Networks Implemented**	
	1	***2***
Logic	75%	79%
BRAM	27%	53%
Speed	199 MHz	198 MHz
FPGA	Virtex II Pro	Virtex 4

Table III shows the overall runtime of the FPGA based implementations and their timing breakdowns. The “Data in time” is the time to read the input image from the hard drive and write it into the FPGA BRAMs. This time is higher for network 2 as the image size for this network is larger. The “Data out time” is the time to read the index of the level 2 neuron that fired (representing the image classification) and to read the simulation cycle count needed for recognition. This time is the same for both networks. The FPGA compute time measure starts when the FPGA is signaled to start processing (after the input image is loaded) and ends when the FPGA signals the AMD processor that recognition has taken place.

TABLE III. Hardware-Accelerated Timing Breakdown

Timing Metrics (ms)	**Networks Implemented**	
	1	***2***
Data in time	0.015	0.105
Data out time	0.003	0.003
FPGA Compute time	0.014	0.082
Total Runtime	0.032	0.190

Table IV compares the runtime of the FPGA accelerated implementation with a fully software implementation for both networks. Speedups of approximately 3.3 times and 8.5 times are seen for networks 1 and 2 respectively. The speedup can be attributed to multiple PEs operating in parallel and to having several neurons processed in parallel through the 23 stage pipeline in each PE. The Virtex 4 implementation provides a higher speedup than the Virtex II Pro implementation primarily because the former has more PEs on chip.

TABLE IV. Performance Measures

Performance Metrics	**Networks Implemented**	
	1	***2***
Software time (ms)	0.105	1.613
Hardware-Accelerated time (ms)	0.032	0.190
Speedup	3.281	8.489

VII. Conclusions

In this paper we implemented a character recognition algorithm based on the Izhikevich spiking neuron model on two types of FPGAs. We developed a modularized PE to evaluate a large number of Izhikevich spiking neurons in a pipelined manner. This PE based design was easily scalable to larger FPGAs. Two network sizes were implemented and showed significant speedups over equivalent software implementations (approximately 3.3 times for the 24×24 pixel image network on a Virtex II Pro and approximately 8.5 times for the 96×96 pixel image network on a Virtex 4 over a 2.2 GHz AMD Opteron core). Our results indicate that FPGAs are suitable for large scale Izhikevich model based cortical simulations.

As future work we plan to further optimize the PEs presented in this study. For example, overlapping the data I/O with the computations on the PEs would have almost doubled the speedups achieved. We also plan to examine the performance and scalability of the PE based design onto newer FPGAs. These would allow for more PEs to be implemented and enable them to run at higher speeds. Finally, we plan to examine the acceleration of larger, more biologically realistic models (such as models with higher connectivity between the neurons) and the training of such models.

Appendix I
Neuron Model Parameters

Excitatory neurons: a=0.02, b=0.2, c= -55, d=4; Inhibitory neurons: a=0.06, b=0.22, c= -65, d=2; time step= 1 ms.

Acknowledgment

This work was supported by grants from the Air Force Research Laboratory (including the AFRL Information Directorate) and a National Science Foundation CAREER award. This work was also supported in part by a grant of computer time from the DOD High Performance Computing Modernization Program at the Naval Research Laboratory.

References

[1] T. Dean, "A computational model of the cerebral cortex," *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pp.938-943, 2005.

[2] R. Ananthanarayan and D. Modha, "Anatomy of a cortical simulator," *Proceedings of ACM/IEEE conference on Supercomputing (SC'07),* 2007.

[3] H. Markram, "The blue brain project," *Nature Reviews Neuroscience*, vol. 7, pp. 153–160, Feb. 2006.

[4] C. Johansson and A. Lansner, "Towards cortex sized Artificial Neural systems," *Neural Networks*, vol. 20, pp. 48-61, Jan. 2007.

[5] Q. Wu, P. Mukre, R. Linderman, T. Renz, D. Burns, M. Moore and Qinru Qiu, "Performance optimization for pattern recognition using Associative Neural Memory," *Proceedings of the 2008 IEEE International Conference on Multimedia & Expo*, pp. 1-4, Jun. 2008.

[6] J. Rickman, "Roadrunner supercomputer puts research at a new scale," Jun. 2008, http://www.lanl.gov/news/index.php/fuseaction/home.story/story_id/13602.

[7] M. M. Khan, D. R. Lester, Luis A. Plana, Alexander D. Rast, X. Jin, E. Painkras, and Stephen B. Furber, "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," In International Joint Conference on Neural Networks (IJCNN), pp. 2849-2856, 2008.

[8] D. B. Strukov, G. S. Snider, D. R. Stewart, R. S. Williams, "The missing memristor found," Nature, 453, pp. 80-83, 2008.

[9] L. O. Chua, "Memristor – the missing circuit element," IEEE Transactions on Circuit Theory, 18, pp. 507-519, 1971.

[10] B. Linares-Barranco and T. Serrano-Gotarredona, "Memristance can explain Spike-Time-Dependent-Plasticity in Neural Synapses," [Online], Available from Nature Proceedings, 2009, http://hdl.handle.net/10101/npre.2009.3010.1

[11] C. Gao and D. Hammerstrom. "Cortical Models Onto CMOL and CMOS—Architectures and Performance/Price," *IEEE Transactions on Circuits and Systems I*, 54(11), pp. 2502-2515, Nov. 2007.

[12] H. Shayani, P.J. Bentley, and A. M. Tyrrell, "Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA", in NASA/ESA Conference on Adaptive Hardware and Systems 2008 (AHS '08), pp. 236-243, Jun. 2008.

[13] A. Upegui, C.A. Pena-Reyes, E. Sanchez, "A hardware implementation of a network of functional spiking neurons with hebbian learning," in International Workshop on Biologically Inspired Approaches to Advanced Information Technology 2004 (BioAdit 04), pp. 399-409, Jan. 2004.

[14] A. Cassidy, S. Denham, P. Kanold, and A. G. Andreou. "FPGA Based Silicon Spiking Neural Array," in IEEE International Workshop on Biomedical Circuits and Systems (BIOCAS'2007), Montreal, pp. 75-78, Nov. 2007.

[15] M. Pearson, I. Gilhesphy, K. Gurney, C. Melhuish, B. Mitchinson, M. Nibouche, and A. Pipe, "A real-time, FPGA based, biologically plausible neural network processor," in International Conference of Artificial Neural Networks, pp. 1021-1026, 2005.

[16] E.Izhikevich, "Which Model to Use for Cortical Spiking Neurons?" *IEEE Transactions on Neural Networks*, vol. 15, pp. 1063-1070, 2004.

[17] E. M. Izhikevich, "Simple Model of Spiking Neurons," *IEEE Trans. Neural Networks*, vol. 14, pp. 1569-1572, Nov., 2003.

[18] M. La Rosa, E.Caruso, L. Fortuna, M. Frasca, L. Occhipinti, and F. Rivoli, "Neuronal dynamics on FPGA: Izhikevich's model,' *Proceedings of the SPIE*, vol. 5839, pp. 87-94, 2005.

[19] L. Fortuna, M. Frasca, and M. La Rosa, "Emergent Phenomena in Neuroscience," in *Advanced Topics on Cellular Self-Organizing Nets and Chaotic Nonlinear Dynamics to Model and Control Complex Systems*, Vol. 63, R. Caponetto, Ed. Hackensack, NJ, World Scientific Publishing Company, pp.39-53, 2008.

[20] M. Mokhtar, D. M. Halliday, and A. M. Tyrrell, "Hippocampus-Inspired Spiking Neural Network on FPGA," *Proceedings of the 8th International Conference on Evolvable Systems: From Biology to Hardware*, pp. 362-371, 2008.

[21] M. A. Bhuiyan, R. Jalasutram, and T. M. Taha, "Character recognition with two spiking neural network models on multi-core architectures," in IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing, Nashville, Tennessee, pp. 29-34, 2009.

[22] W. Gerstner, and W. Kistler, *Spiking Neuron Models, Single neurons, Populations, Plasticity*, Cambridge University Press, 2002.