Proceedings of the 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control - CESCIT 2012
3-5 April 2012. Würzburg, Germany

FPGA implementation of Spiking Neural Networks

A. Rosado-Muñoz, M. Bataller-Mompeán, J. Guerrero-Martínez

Dpt. Electronic Engineering. School of Engineering. University of Valencia.
Avda. Universitat s/n. 46100 Burjassot. Valencia. SPAIN (e-mail: Alfredo.Rosado@uv.es).

Abstract: Spiking Neural Networks (SNN) have optimal characteristics for hardware implementation. They can communicate among neurons using spikes, which in terms of logic resources, means a single bit, reducing the logic occupation in a device. Additionally, SNN are similar in performance compared to other neural Artificial Neural Network (ANN) architectures such as Multilayer Perceptron, and others. SNN are very similar to those found in the biological neural system, having weights and delays as adjustable parameters. This work describes the chosen models for the implemented SNN: Spike Response Model (SRM) and temporal coding is used. FPGA implementation using VHDL language is also described, detailing logic resources usage and speed of operation for a simple pattern recognition problem.

Keywords: neural network models, hardware synthesis, FPGA, VHDL, spiking neural network.

1. INTRODUCTION

Traditionally, *Artificial Neural Networks* (ANN) are those with sigmoidal neuron (Tommiska, 2003). Deep studies on sigmoidal Neural Networks have been carried out during the last decades. However, there exist other neural structures that we could also call "artificial" such as SNN. Spiking Neural Networks are inspired in the biological neural system, trying to obtain a computational system able to replicate the behavior of biological neurons for both information transmission among neurons and internal neuron signal processing. Nowadays, the behavior of the biological neural system is well known (Arbib et al., 2002) and thus, a modeling can be done. One of the most interesting properties of SNN are the delays involved in the system. Both weights and delays can be adjusted to provide adequate behavior of the network. Traditional ANN are based on weights, but SNN also include the delay as a parameter. Depending on the length of a synapse, the signal delay from one neuron to the next is different. This fact gives more flexibility and introduces more parameters for training SNN.

Different applications have been developed using SNN, image recognition and operation (Meftah et al., 2008), RGB value interpretation (Meftah et al., 2008), retinal coding and image processing (Perrinet, 2008), filtering for image processing (Wu et al., 2008), fast and adaptive image processing with multi-view pattern recognition (Wysoski et al., 2008), and many other applications in different fields.

In general, most of the aforementioned applications were developed under computer systems and offline applications. Since Spiking Neural Networks are an implementation of real biological neural system, they can be applied to almost every field, for instance, its wide usage can be found in robotics for machine control movement as in (Floreano et al., 2001; Hagras et al., 2009). Other applications including navigation tasks or obstacle avoidance by robots (Roggen et al., 2003) have also been developed. For this reason, hardware implementation for SNN is an interesting topic because real time operation can be achieved and embedded hardware systems can be created for on-line tasks, enlarging the number of applications where SNN can be applied. In some cases, a custom VLSI chip (Vogelstein et al., 2005) was used.

This work describes the hardware implementation of a SNN. A general approach is used and different SNN structures can be generated by parameterizing the code, providing flexibility to adapt the hardware to specific needs. Using this approach, the same code with the adequate parameters could be used for different applications, allowing a fast prototyping system for a hardware system. This paper is not focused on the learning algorithm for weight update, typically, Spike-Timing Dependent Plasticity (STDP) algorithm (Song et al., 2002) is used as a learning rule for the network training process (Ponulak et al., 2010), (Benuskova et al., 2007) or other supervised techniques as backpropagation (Schrauwen et al., 2006). If weights must be changed during the hardware operation life, FPGA device must be reprogrammed. The proposed VHDL code includes a convenient module where weights can be changed easily. In previous works, a software system is also defined to interface with the proposed hardware design (Rosado et al., 2011).

The proposed system is tested for a pattern classification task in a 3x3 black and white image. Some similar application of pattern recognition is also described in

(Booji, 2004). The paper is structured as follows: section 2 introduces the neural model and coding used. Section 3 describes the hardware structure of the proposed system. Section 4 gives the results of testing and its implementation (occupation and performance). Finally, section 5 concludes the paper and points some guidelines for future work.

2. SPIKING NEURONS AND INTERCONNECTIONS

2.1. Spike Response Model - SRM

Several models exist to obtain the internal behavior of a neuron (Gerstner et al., 2005). In this paper, the Spike Response Model (SRM) is chosen due to its similarity to a biological neuron as recent research studies support (Paugam-Moisy et al., 2009; Booji, 2004) The main characteristic of a spiking neuron is the membrane potential, normally staying at a resting value of -75mV. In case of two connected neurons, we refer to a transmitting neuron as a presynaptic neuron while a receiving one is the postsynaptic neuron. When a new spike arrives to a postsynaptic neuron it generates a change on its membrane potential called Post Synaptic Potential (PSP). This change can increase (excitatory PSP) or decrease (inhibitory PSP) the neuron membrane potential. It is not clear why some neurons are excitatory and others inhibitory, but it is commonly assumed that both types of neurons are present in a complex SNN. A general PSP function is defined by (1), where a weight factor ω for calculating the PSP value is applied to every connection (Fig. 1) in order to provide more significance compared to other neurons.

$$PSP(x) = w \cdot \left(e^{\left(-\frac{x}{4}\right)} - e^{\left(-\frac{x}{2}\right)} \right) \qquad (1)$$

A postsynaptic neuron increases its membrane potential up to a threshold value υ; then, the neuron fires an output spike and enters in a refractory period. During the refractory period, the neuron cannot react on any new incoming spike. The refractory period can be described by (2) (Booji, 2004):

$$\eta(x) = -\upsilon \cdot e^{\left(-\frac{x}{20}\right)} \qquad (2)$$

Being $t_i^{(f)}$ the time when a spike is fired by a presynaptic neuron (indexed by i), this spike is changing the potential of a postsynaptic neuron j at time t. The time difference between these two events is $t-t_i^{(f)}$, and, including the delay of the synapse (denoted as d_{ji}), the travelling time between two neurons for a spike is obtained according to (3).

$$\Delta t_{ji} = t - t_i^{(f)} - d_{ji} \qquad (3)$$

In order to obtain the model for the entire neuron behavior, when a sequence of spikes $F_i=\{t_i^{(g)},\ldots,\ t_i^K\}$ arrives to a neuron j, the membrane potential changes according to the PSP function and refractory period, and thus, a spike train is propagated by neuron j as $F_j=\{t_j^{(f)},\ldots,\ t_i^N\}$. The composed equation for j neuron potential u_j as a function of time is obtained by (4).

$$u_j(t) = \sum_i^K \sum_{t_i^{(g)} \in F_i} PSP(t - t_i^{(g)} - d_{ji}) + \sum_{t_j^{(f)} \in F_j} \eta(t - t_j^{(f)}) \qquad (4)$$

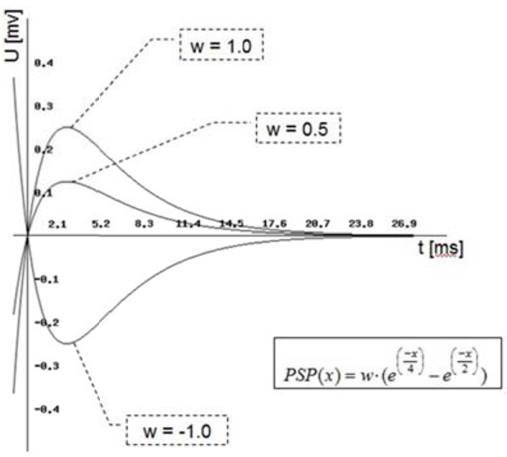

Fig. 1. Postsynaptic potential function – PSP with weight dependency.

The membrane potential of one postsynaptic neuron is described in (4), where the influence of every neuron input is added up and also refractory period is taken into consideration.

2.2. Neural Coding

When a stimulus is presented to the network input, input neurons generate a spike or a series of spikes (spike trains) connected with this stimulator; if the stimulus is changed, the reaction of neurons change and different spike trains are generated. If this response is recorded in a time window, spike raster plots are typically used to evaluate spikes along time.

There exist different coding information methods (Goldberg et al., 2007; Gerstner et al., 2002). Using temporal coding, the input neuron fires a spike regularly every time interval while the input stimulus is active (Paugam-Moisy et al., 2009). Other common coding techniques are Gaussian Receptive Fields (GRF) (Beňuskova, 2009; Fijałkowski, 2009). In this case, the proposed method is a modification of temporal coding where the input neuron does not calculate any membrane potential, the input neuron generates a spike every 50 milliseconds (tested to be the most adequate) while the digital input is active (value ‘1’).

In a simple pattern recognition example, we have black and white patterns received in a 3x3 pixel image and thus, nine input neurons according to Fig. 2. While a pixel is black (active ‘1’), the neuron corresponding to this pixel fires a spike regularly within every time period, if pixel is white, the neuron is not active (Fig. 3). Fig. 3 also shows the synapse delay (bottom of figure) where spikes are delayed in a different value depending on the corresponding input spike.

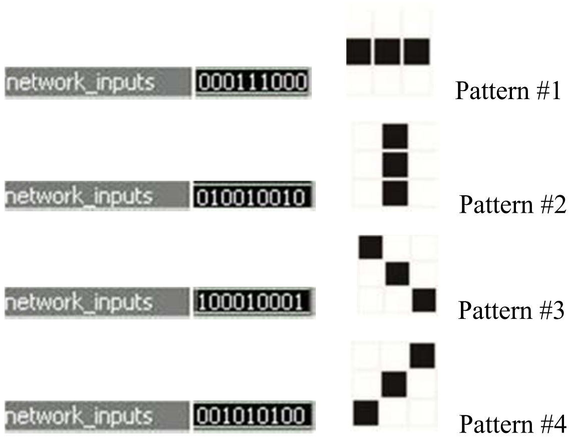

Fig. 2. Input neuron activation values according to different patterns in a black and white 3x3 pixel image.

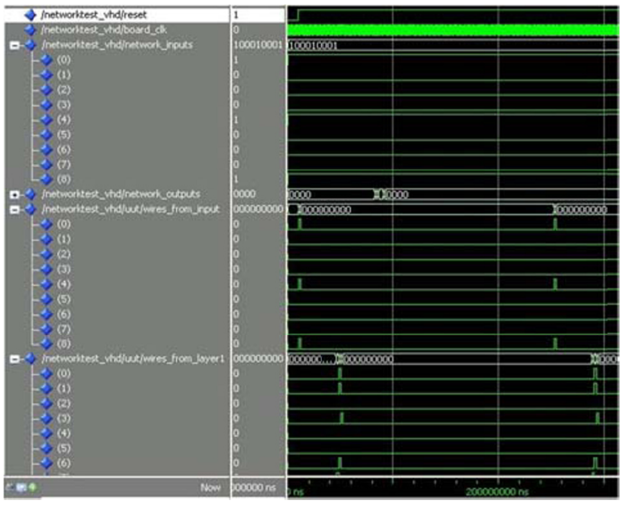
/networktest_vhd/reset
/networktest_vhd/board_clk
/networktest_vhd/network_inputs
/networktest_vhd/network_outputs
/networktest_vhd/uut/wires_from_input
/networktest_vhd/uut/wires_from_layer1
Now
200000000 ns

Fig. 3. Temporal spike coding for a set of inputs.

2.3. Neuron Activation.

A single neuron is connected to all previous neurons. Thus, multiple spikes arrive to the neuron at different times. For every spike, a PSP function is launched, changing the potential in the neuron. Fig. 4 shows the result of arriving different spikes at different time, being the potential increased for each spike; when the potential is above a threshold, the spike is generated and the refractory period activated.

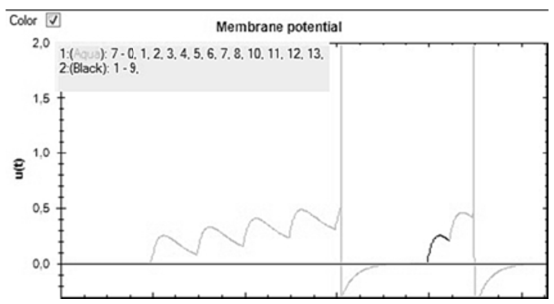

Fig. 4. Membrane Potential graph.

3. HARDWARE IMPLEMENTATION

The SNN structure has been developed in VHDL language targeting FPGA devices. A hierarchical structure approach was used, allowing high modularization and parameterization. A *global.vhd* package holds types used by the entire structure, parameters for the network such as the number of neurons in every layer, synapses delays, and every other constants or variables modification required by the appropriate hardware generation. Using this approach, the design can be modified to implement different SNN structures by only changing one parameter file. The entire neural structure is generated with *generic* statements and *for* loops to create instances of every network element and map all ports to provide appropriate connection. Two clock signals are defined, one clock is defined for a 1ms period the time base for spike transmission, and a second clock for computational calculations required before a new spike evaluation which might vary depending on the target and type of implementation (typically in the order of MHz).

The input neuron is responsible for generating a spike train when the input is active; the time separation between spikes is defined by a parameter defined in the global package.

The synapse component is designed to delay a spike travelling from one neuron to the next. It has been implemented using a one bit FIFO. The FIFO length corresponds to the number of clock cycles that the synapse must delay the spike, i.e. when a presynaptic neuron fires a spike a synapse input bit is set to 1 and this bit is shifted along an internal vector every millisecond. When this bit leaves the FIFO, the synapse delay is met. Fig. 5 shows a simple diagram containing the basic modules of the proposed SNN structure: input neurons, synapses and internal neurons.

A single neuron hardware model was developed and can be seen in Fig. 6. There exists a PSPgenerator component for every input to the neuron. When a spike arrives to the input, the *PSPgenerator* module input is set to logic '1', then, the *PSPgenerator* module starts to read memory-stored PSP function values defined in the global package. The values generated for each of the *PSPgenerator* modules are added by two components: *Potential_Mux* and *AddUnit,* responsible for adding all the PSP values. The sum of PSP function values from every generator is compared to the threshold by *PotentialControl* unit. If the sum of values is above the threshold, the output of the neuron module is set to '1' for 1ms, meaning that a spike has been generated. *PotentialControl* unit is also responsible for the refractory period to avoid a new activation during that time.

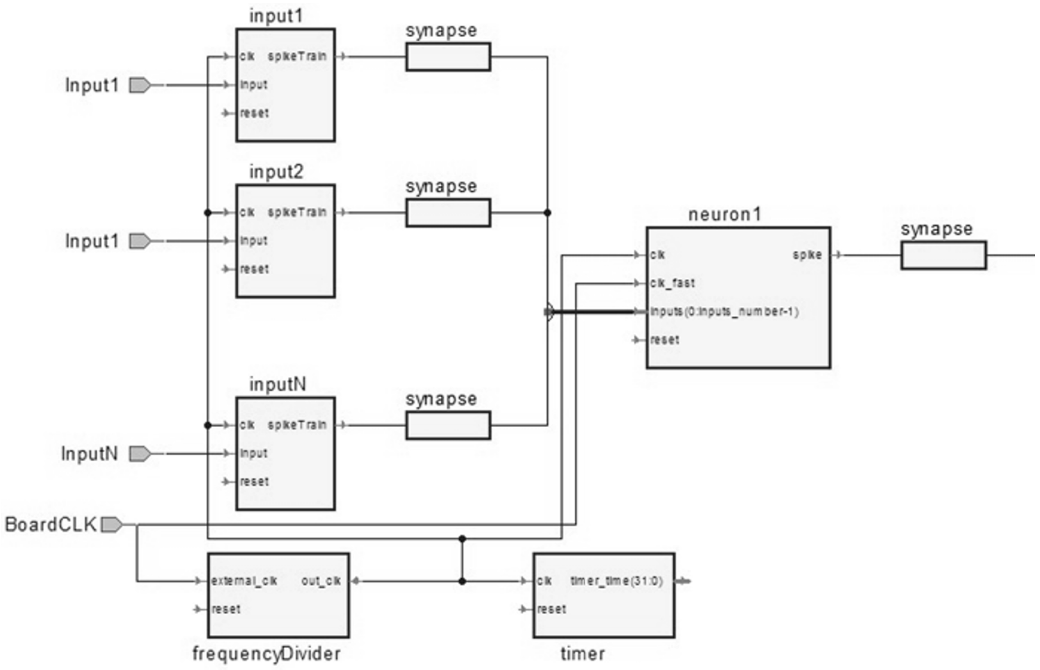

Fig. 5. Spiking neural network example with several inputs and one neuron in the first layer.

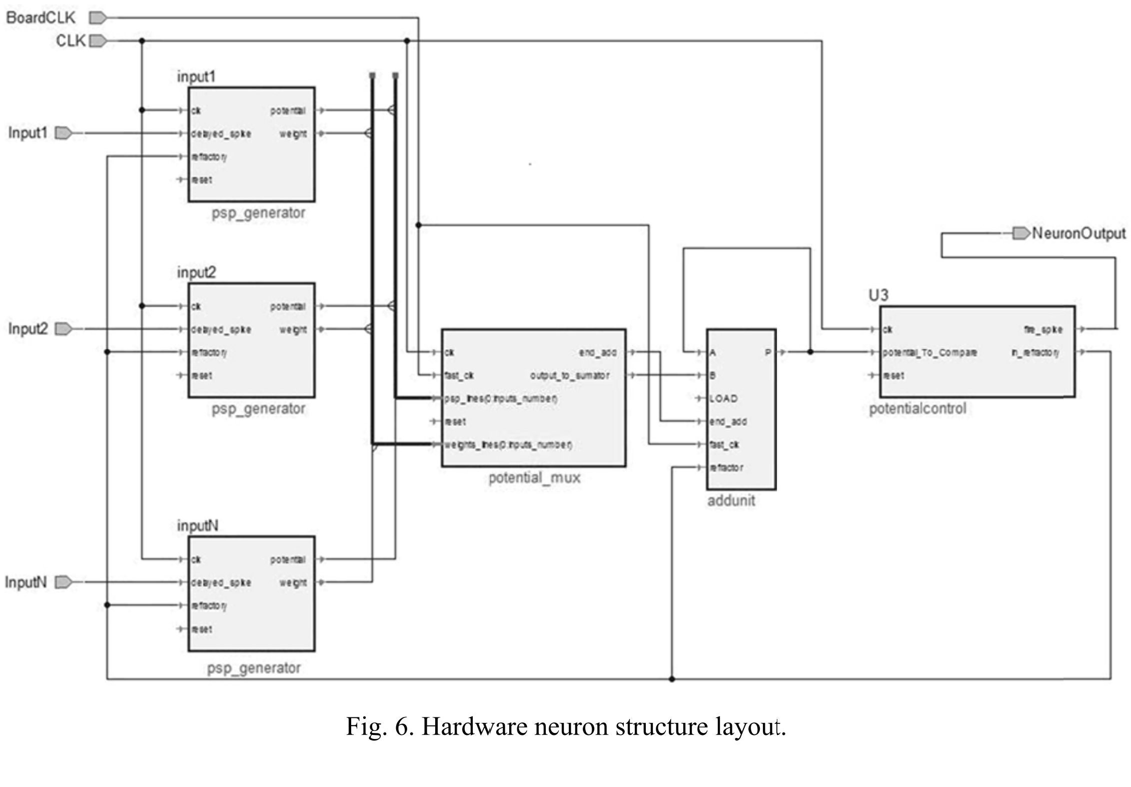

Fig. 6. Hardware neuron structure layout.

4. RESULTS

A simple pattern recognition problem is proposed, a black and white 3x3 pixel image is fed and patterns shown in Fig. 2 must be classified. The SNN associates one output neuron with one pattern. As the best result, only one output neuron should fire when a particular pattern is presented to the input. However, it is common that more than one output neuron fires, thus, the so-called winner-takes-all rule is used, i.e. the first firing neuron after the input stimulus is present is taken as the winner. A 9-9-4 SNN structure is defined, consisting on nine input neurons, nine neurons in the hidden layer and 4 outputs. Fig. 7 shows the classification results, showing a different output being active first depending on the input pattern

The designed VHDL code is fully synthesizable on FPGA, as an example, the Xilinx Spartan3 XC3S4000 device is used. For comparative purposes, table 1 shows the resources used in different SNN structures, including the SNN used in the example (bold letters).

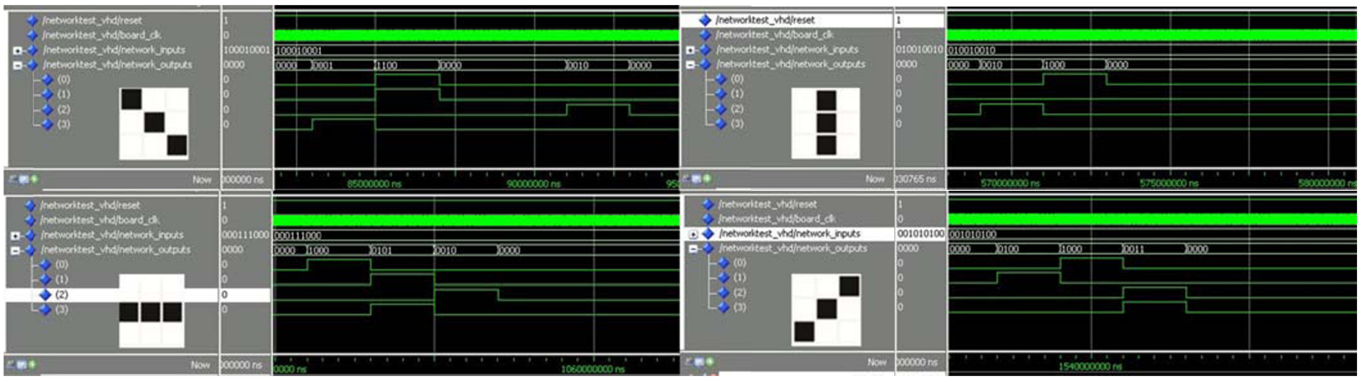
/networktest_vhd/reset
/networktest_vhd/board_clk
/networktest_vhd/network_inputs
/networktest_vhd/network_outputs
(0)
(1)
(2)
(3)
Now

Fig. 7. Classification results for 3x3 pixel patterns.

SNN Structure	Slices Utilization	LUTs Utilization
Single neuron	202	378
2-4-1	2268	4180
32-4-16	11340	21059
9-9-4	**7860**	**13885**

Table 1. FPGA resource utilization according to different synthesized structures. For the SNN structure, numbers are: InputLayer - HiddenLayer1 – OutputLayer

Logic occupation remains in a low value compared to the total size available in modern FPGA. It is estimated that an SNN with more than 200 neurons could fit easily in an FPGA device, being able to solve complex application problems. Concerning performance, 20 MHz clock rate can be achieved for the fast clock, providing enough time to perform all required computation considering that the next spike sample time is 1ms.

5. CONCLUSIONS

Implementation of SNN in an FPGA was done. Generic VHDL code was developed in order to create different SNN structures by just modifying a package file containing all parameters.

Although logic resources used were optimized, FPGA occupation could be further optimized. In the proposed design, the *psp_generator* component is replicated for every instantiation. It is synthesized as a single ROM unit. For a network with N input neurons, M neurons in hidden layer and K neurons in output layer the total number of *psp_generator* components is equal to:

$$\rho = N \cdot M + M \cdot K \tag{5}$$

A solution to reduce utilization is to design a *psp_generator* component able to calculate the PSP function during one clock cycle. However, this solution introduces some limitation to PSP function characteristic due to difficulties with calculations in hardware modeling. Some other solutions must be investigated. If a shared PSP generator could be developed, logic occupation could be improved.

Additionally, clock speed of operation is enough for spike data rate.

REFERENCES

Arbib, M.A(editor) and Others (2002). *The Handbook of Brain Theory and Neural Networks: Second Edition*, MIT Press, Cambridge, USA

Benuskova, L., and Abraham, W.C. (2007) *STDP rule endowed with the BCM sliding threshold accounts for hippocampal heterosynaptic plasticity*. Journal of Computational Neuroscience vol 22, no. 2 pp:129-133

Booij, O. (2004). *Temporal Pattern Classification using Spiking Neural Networks*, Master's thesis. University of Amsterdam, Amsterdam, The Netherlands

Fijałkowski, A.B. (2010). *Software Environment for Spiking Neural Network Design and FPGA implementation*, Master's thesis. University of Valencia, Spain

Floreano, D., and Mattiussi, C. (2001) *Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots* in *Evolutionary Robotics. From Intelligent Robotics to Artificial Life*. Book Series Title: Lecture Notes in Computer Science. Editor: Gomi, Takashi. Springer Berlin/Heidelberg, Vol 2217. pp. 38-61

Gerstner, W. and and Kistler,W.M. (2002). *Spiking Neuron Models*, Cambridge University Press, UK. http://lcn.epfl.ch/~gerstner/BUCH.html

Goldberg, D.H. and Andreou, A.G. (2007). *Distortion of Neural Signals by Spike Coding*, Neural Computation. Vol. 19, No. 10, pp. 2797-2839

Hagras, H., Pounds-Cornish, A., Colley, M., Callaghan, V. and Clarke, G. (2004). *Evolving Spiking Neural Network Controllers for Autonomous Robots*, Proceedings of ICRA'2004. pp.4620-4626

Meftah, B., Benyettou, A., Lezoray, O. and QingXiang, W. (2008). *Image Clustering with Spiking Neuron Network*, IEEE International Joint Conference on Neural Networks (IJCNN). pp 681-685

Paugam-Moisy, H. and Bothe, S. (2009). *Computing with Spiking Neuron Networks*, Handbook of Natural Computing, Springer-Verlag

Perrinet, L.U. (2008). *Adaptive sparse spike coding: applications of neuroscience to the compression of natural images*, Optical and Digital Image Processing Conference. Proceedings of SPIE Volume 7000, pp. 70000F

Ponulak, F. and Kasiński, A. (2010). *Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence*

Learning, Classification, and Spike Shifting, Neural Computation. Vol. 22, No. 2, pp 467-510

Roggen, D., Hofmann, S., Thoma, Y. and Floreano, D. (2003) *Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot*. NASA/DoD Conference on Evolvable Hardware (EH'2003), pp. 189-198.

Rosado, A., Fijałkowski, A.B., Bataller, M., Guerrero J., (2011) *FPGA implementation of Spiking Neural Networks supported by a Software Design Environment.* IFAC 2011 World Congress. Milan, Italy. vol. 18, part 1. pp 1934-1939.

Schrauwen, B. and van Campenhout, J. (2006). *Backpropagation for Population-Temporal Coded Spiking Neural Networks*, International Joint Conference on Neural Networks (IJCNN). pp 1797-1804

Song, S., Miller, K.D. and Abbott, L.F. (2000). *Competitive Hebbian learning through spike-timing dependent synaptic plasticity*, Nature Neuroscience, volume 3 no. 9, pp 919-926

Tommiska, M.T. (2003).. *Efficient digital implementation of the sigmoid function for reprogrammable logic,* IEE Proc.-Comput. Digit. Tech. Vol. 150, no. 6, pp 403-411

Vogelstein, J.R., Culurciello, E., Mallik, U. and others, (2005). *Saliency-Driven Image Acuity Modulation on a Reconfigurable Silicon Array of Spiking Neurons*, Cambridge, MA: MIT Press, 2005, 17, pp.1457–1464

Wu, Q.X., McGinnity, T.M., Maguire, L.P., Belatreche, A., and Glackin B. (2008) *Processing visual stimuli using hierarchical spiking neural networks*, Neurocomputing, vol 71, pp 2055–2068.

Wysoski, S.G., Benuskova, L., and Kasabov N. (2008), *Fast and adaptive network of spiking neurons for multi-view visual pattern recognition*, Neurocomputing, vol. 71, pp 2563–2575.