Adaptive Boolean Logic Using Ferroelectrics Capacitors as Basic Units of Artificial Neurons

Alan P. O. da Silva[1], Cicília R. M. Leite[2], Ana M. G. Guerreiro[3], Carlos A. Paz de Araujo[4] and Larry McMillan[5]

[1,3]*Federal University of Rio Grande do Norte*
[2] *State University of Rio Grande do Norte And College of Science and Technology Mater Christi*
[4,5]*Symetrix Corporation*
[1,2,3]*Brazil*
[4,5]*USA*

1. Introduction

Neural networks are being investigated as a computational paradigm in many fields of artificial intelligence.
The pioneers of cybernetics were clearly inspired by neurology and the current knowledge of the human brain to develop the architectures of modern computing machines. The evolution has given the brain very distinctive capabilities of learning, distributed memory, computation, generalization, robustness, and the capability of interpretation of imprecise and noisy information, etc., not present in Von Neumann computers.
Neuroengineers have come up with the idea of using Artificial Neural Networks (ANNs) massively parallel computing machines inspired by the biological nervous systems. However, ANNs have a very different approach and computing paradigm, where learning from examples and learning from iteration replaces our common idea of programming. These models achieve good performance via massively parallel networks composed of generally nonlinear computational elements, referred to artificial *neurons*. Synaptic weights are associated with each neuron connection between neurons. In the case of classical ANNs, the activation potential and the synaptic weights are analogous, respectively, to the firing rates and the strength of the synapse in biological neurons which are arranged in layers.
The first, very simplified model, mathematical model of a neuron operating in an all-or-none fashion: the Threshold Logic Gate (TLG). It did not take very long for a hardware implementation to be developed. Since then a lot of researh have been developed with the aim of implementing artificial neurons in hardware and construct intelligent systems On-Chip.
Many theoretical results have been shown that threshold logic circuits are more powerful and/or efficient than Boolean circuits (Beiu et al., 2006). Exploiting the principle for electronics circuits may reduce the number of transistors and interconnections (Shibata and Ohmi, 1991). Output-wired threshold gates working in the classical above threshold domain have been implemented, too. A lot of work has been done in this area. Thinking on this scenario, this chapter shows a implementation of the boolean logic with artificial neuron type model with a

Ferroelectric (Fe) capacitor as its basic unit. The Fe Capacitors was chosen because it have been embedded into LSIs as Ferroelectric Random Access Memory (FeRAM) and their reliability data have been accumulated. The capacitors are high impedance device, and it is an advantage for low power consumption, besides the configuration can be changed after packaging. The FeCapacitor uses the phenomenon of the hysteresis loop as the activation function for the neuron model. The model performs a weighted sum of the inputs and applies it the non-linear activation function generated by a single side of the hysteresis. Changing the weights, it is possible to reconfigure the gate easily. Ones tries to show a new way of implementing a neuron, without using transistors.

We developed the perceptron model with the FeCapacitor, that we called here FePerceptron, and we also developed the FeSpiking Neuron model, that is an extended model for the spiking neuron (Guerreiro et al., 2008) using the FeCapacitor as its basic unit for the activation function. Both models is going to be simulated and tested in Matlab. The models were used to simulated all logical gates and synthesis of several digital circuits as D-flip-flop, shif-register, full adders, and a simple CPU with the Spiking FeNeuron. The Spiking FeNeuron is developed because it can simulated all logical gates, inclusive the XOR gate with single neuron, which is impossible with a single Perceptron.

Because of the simplicity, we started the hardware implementation with the FePerceptron in an Field Programmable Gate Arrays (FPGA). The FeCapacitor is developed as its basic unit and can be used in any neuron model as activation function. We used the DSP builder of Altera to generate the model. The DSP Builder Signal Compiler block reads Simulink Model Files developed (.mdl) that are built using DSP Builder blocks and generates the VHDL files. This is the first step of a work to implement in hardware the neuron using FPGAs simulating the desired hardware, bringing a second degree of reconfigurability to the FPGAs with the adaptive boolean logic CPU.

This chapter is organized as follow, first in section 2 we give an overview of several developed works and researches that were done simulating and implementing neural networks in hardware. Section 3 shows the development of the mathematical model of the FeCapacitor. The FeCapacitor model is used as activation function to the Perceptron model generating the FePerceptron and to the Extended Spiking Neuron model generating the FeSpiking Neuron Model as shown in section 4. The simulations in Matlab of both models are shown in section 5 with the realization of boolean logic gates and adaptive boolean circuits. The section 6 is responsible to shown the FePerceptron model developed in an FPGA. Finally, conclusion are presented in section 7.

2. History of neuron models, simulations and implementation of neural networks in hardware

The hardware implementations means and introduces for example computational errors, degradation of learning and lack of accuracy in results. This are some issues that have been studied and address and explored in many researches. These include digital (Bermak and Martinez, 2003; Kung, 1992; Lenne, 1995), analog (Brown et al., 2004; Mead, 1989), hybrid (Lehman et al., 1996; Schmid et al., 2004), FPGA based (Nedjah and Mourelle, 2007; Rak et al., 2009; Schrauwen and D'Haene, 2005), and (non-electronic) optical implementations (Moerland, 2007; Tokes et al., 2000).

In spite of all the difficults to implement ANN in hardware a lot of research have been done. There is a lot of needs in real-world applications. Some examples are: optical character recognition, robotics, voice recognition, adaptive filters, image analysis, finger print feature

extration, acoustic sound recognition, olfactory sensing, traffic monitoring, experiments in high energy physics (Lamela and Ruiz-Llata, 2005), and adaptive control.
There are indeed several surveys which have appeared in the past. There is this one (Janardan and Indranil, 2010) that has done a survey which includes most of all works concerning Hardware Neural Networks (HNN). Here, we only give a small overview over what was done in the past based on (Janardan and Indranil, 2010), so the importance of the HNN can be noticed, and seen that a lot of research has already been developed. Some early references on the VLSI implementations can be found on (Mead, 1989) and (Glesner and Poechmueller, 1994). An overview on neurocomputers up to the 90's, built from accelators boards, general purpose hardware, and neurochips can be found on (Heemskerk, 1995). Digital implementations with custom processors can be found (Ienne, 1996).
Sundararajan and Saratchandran (1998) discuss in detail various parallel implementation aspects of several ANN models using various hardware architectures. Zhu and Sutton (2005) survey FPGA based implementations of ANNs discussing different implementation techniques and design issues. Smith also discuss and survey digital and analog VLSI implementations approaches in neural model.
One of the latest surveys with specific focus to commercially available hardware can be found on Dias et al. (2004). Neurofuzzy hardware systems is discussed concerning aspects of various tecnologies of hardware implementations and software co-design techniques. Implementations of Spiking Neural Networks is provided by (Schrauwen and D'Haene, 2005). Valle (2005) presents various approaches to built smart adaptive devices.
There still some topics as hardware friendly learning algorithms, as pertubation learning (Jabri and Flower, 1991), constructive learning (Smieja, 1993), cascade error projection learning (Duong, 1995; Duong and Stubberud, 1995), and local learning (Chen et al., 2000). Some HNN based on Multilayer Perceptron (MLP) (D'Acierno, 2000; Kumar et al., 1994), radial basis function (Fakhraie et al., 1994; Verleysen et al., 1994; Yang and Paindavoine, 2005), and Neocognition (Patnaik and Rao, 2000) and neurocomputers(Glesner and Poechmueller, 1994; Strey and Avellana, 2000).

3. The mathematical model of ferroelectrics capacitor

There are two areas of research to be investigate, the physically based models and the behavioral models. In our work, the bahavioral modeling was chosen because it does not required a detailed knowledge of ferroelectric theory; it only requires a careful observation of the ferroelectric capacitor behavior from the circuit point of view.
A lot of research have been done in attempts to models the behavior of a ferroelectric capacitors (for a review, see: (Sheikholeslami and Gulak, 1997) since they were introduced as storage elements in integrated nonvolatile memory applications.
In this paper, the Mathematical Model (Miller et al., 1991) is used that introduces a closed form mathematical equation for the hysteresis loop.
Based on the saturability of the switching polarization and the symmetry of the hysteresis loop, the mathematical model approximates the saturated polarization loop with two hyperbolic functions:

$$P_{sat}^{+}(E) = P_s tanh[\frac{E - E_c}{2\delta}] \tag{1}$$

and,

$$P^{-}_{sat}(E) = -P^{+}_{sat}(-E) \tag{2}$$

where $P^{+}_{sat}(E)$ and $P^{-}_{sat}(E)$ represent the polarization corresponding to the positive and negative going branches of the hysteresis loop, respectively. P_s and E_c are the saturation polarization and the coercive field extracted from an actual hysteresis loop. With the fixed P_s and E_c, δ is uniquely specified by P_r, the remanent polarization, through the following equation:

$$\delta = E_c[ln(\frac{1+P_r/P_s}{1-P_r/P_s})]^{-1} \tag{3}$$

A sketch of the hysteresis loop is done with MATLAB as it is shown in Figure 1. The symmetry with respect to the origin in this Figure is guaranteed by Equation (2).

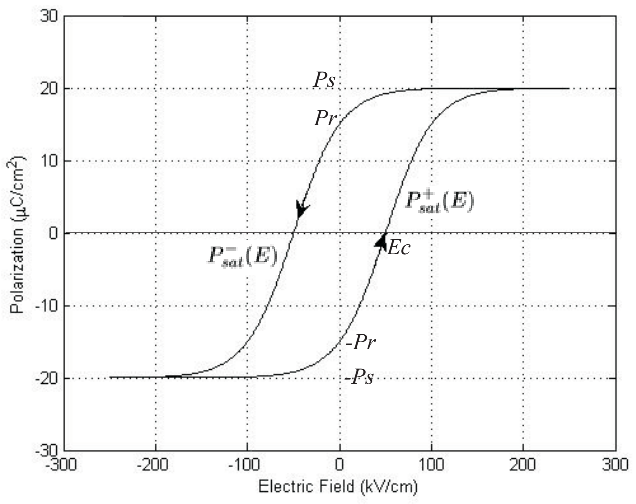


Fig. 1. The hyperbolic tangent functions approximate the Saturated Polarization Loop. The hysteresis loop is plotted considering the values of P_s = 23 $\mu\ C/cm^2$, P_r = 15$\mu\ C/cm^2$, and E_c = 40kV$/cm^2$ borrowed from an actual saturated hysteresis loop.

The Mathematical Model provides a good approach for steady-state analysis of ferroelectric capacitor behavior, and this model is sufficient by now for this work. Future simulations maybe can incorporate the transient analysis and by than mode accurate models can be necessary.

4. The ferroelectrics capacitor as an activation function of artificial neurons

4.1 Perceptron model

The biological neuron is the structure unit of the nervous systems. It considers of a cell body, called soma, axon and several ramifications. These ramifications are called, dendrites. The dendrites conduct action potentials or electrical pulses toward the body cell and conform in large and complicated trees. The dendrites and the soma constitute the input surface of the neuron. The axon consists of a large fiber whose branches form the axonal tree. The axon has a arborization at its terminal. The tips of the branches of the axon are called nerve terminals, boutons or synaptic knobs. The axon acts as a transmission line. The action potential travels along an axonal tree all the way to the nerve terminals. The terminals of the branches make contacts with the dendrites of other neurons. The contacts are called synapses.

The first computational model of the biological neuron was introduced by McCulloch and Pitts (McCulloch and Pitts, 1943) in the 1940s. McCulloch and Pitts merged mathematical logic and neurophysiology to propose a binary threshold unit as a computational model for an artificial neuron operating in discrete time. The output $y_k(t)$ of a neuron is 1 when an action potential is generated, and -1 otherwise. A neuron fires when the effects of inhibitions and excitations are larger than a certain threshold, θ.
In 1958, the American psychologist Rosenblatt proposed a computational model of neurons that he called *The Perceptron* (Rosemblatt, 1958). The essential innovation was the introduction of numerical interconnection weights.
A neuron is an information processing unit that is fundamental to the operation of a neural network (Haykin, 1998). The perceptron model of the neuron has three basic elements (Figure 2):

1. Synapses that are characterized by the strength of the weights;
2. An adder for summing the input signals, weighted by the respective synapses and the threshold of the neuron;
3. An activation function for limiting the amplitude of the output (in this case completely unclocked with the firing event).

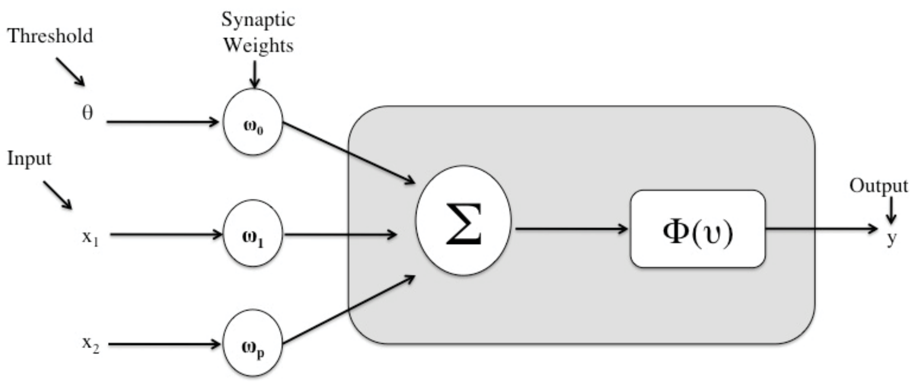


Fig. 2. Perceptron Model

The external threshold, denoted by θ, has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative.
The perceptron consists of a linear combiner followed by a non-linear function, as depicted in Figure 2. The summing node of the neuronal model computes a linear combination of the inputs, x_i, applied to the synapses connections, w_i, and also incorporates an external threshold, θ. The resulting sum, that is, the induced local field, is applied the activation function. The activation function denoted by $\Phi(v)$, defines the output of the neuron in terms of the local field, v. The activation function usually used is a sigmoid. The weights model the synaptic strength, and the output of the neuron models the rate of fire of biological neurons. The threshold can be accounted for in two ways: adding a new input signal fixed at $+1$ or adding the threshold to the linear combination of the input with the weights.
The neuron can be described mathematically as:

$$v = \sum_{j=1}^{p} \omega_j x_j \tag{4}$$

and

$$y = \phi(v + \omega_0\theta) \tag{5}$$

The capacity of learning is one of the most important characteristics of an ANN. Learning is closely related to an improvement of performance of the system. This is achieved by minimizing errors, self-organizing information through correlations of data, and maximizing rewards in a trial-and-error system over time. In practice, learning is achieved by adjustment of the free parameters of the ANN.

4.1.1 The FePerceptron model

Our simplified model is based on the perceptron model, using the following concepts: inputs, synapses strength, adder, activation function and the threshold.
The FeNeuron uses the ferroelectric capacitor as its basic unit. The model (Figure 3) is composed by the input, x_i, by the synaptic weights, w_i, as resistors, combining them linearly. The result is applied to the ferroelectric capacitor performing the output of the model. The phenomenon of the hysteresis loop is used to act as the activation function. It is necessary to use only one side of the hysteresis loop.

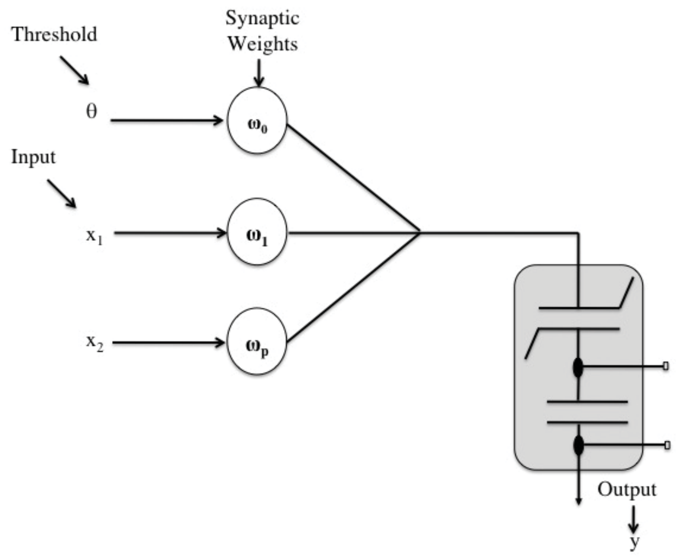


Fig. 3. The FeNeuron Model

The induced local field of the neuron is computed as in Equation (4). The output is described as:

$$y = P^+_{sat}(v + \omega_0\theta) \tag{6}$$

and,

$$P^+_{sat} = P^+_{sat}/P_s \tag{7}$$

The Equation (6) can be re-written as:

$$y = P^+_{sat}(\sum_{j=1}^{p} \omega_j x_j + \theta) \tag{8}$$

The goal of the model in this case is to classify correctly the set of externally applied stimuli, $x_1, x_2, x_3, ..., x_p$ into two classes. In our work, we are treating with Boolean functions with two inputs, thus our network is a single layer, with only one neuron with two inputs plus the threshold and one output.

4.2 Extended Spiking Neuron model

Artificial spiking neurons are biophysical models which try to account for properties of biological neurons by modeling the integrated signal flow through parts of the neuron. The input spike signals, from presynaptic biological neurons are weighted, summed up and passed through a type of leaky integrator. If the membrane potential exceeds a certain threshold, a spike is generated. The pulses arrive at the input synapses in discrete time. The Extended Spiking FeNeuron is a discrete model composed of an input layer, synaps weights, first order recursive filters, a soma, a hard limiter and the membrane potential. The input layer receives the input spikes. The weights are the synaptic strength values. The soma provides a linear combination of the input signal passing through the first order recursive filters and summing with the thresholds coming from the feedback loop. The hard limiter compares the soma potential with the threshold and emits a spike depending on the soma threshold.
The membrane potential is responsible to compute the delay necessary for the neuron to compute. The delay is build up by the summing potential of the input signals and depends on the time delay of the filters and more important on the delay caused by the feedback loop where the dynamic threshold sets the time of firing. We can define a neuron i that receives inputs from presynaptic neurons $j \in \Gamma_j$, where

$$\Gamma_i = \{j/j \ \ presynaptic \ \ to \ \ i\} \tag{9}$$

The discrete model considers a "1" for a incoming spike and "0" for the absence of a spike. The input passes through the first order recursive filters and then is summed up. The Equation 10 describes the action of the first order recursive filters:

$$u_{fij} = e^{-(T/\tau_{ij})} u_{fij}(n-1) + \omega_{ij}(n) x_{ij}(n) \tag{10}$$

where $x_{ij}(n)$ is the input of the synapses, $w_{ij}(n)$ is the synapses weight, T_{ij} is a constant delay of the filter, $u_{fij}(n)$ is the output of the first order recursive filter in a time slice, n.
The dynamic threshold is describe by the following equation:

$$\theta(n) = \vartheta + \theta_1(n) \tag{11}$$

$$\theta_1(n) = p_i(n) r_i(n) \tag{12}$$

where ϑ is the static threshold and $pi(n)$ is the output of the first order recursive filter of the feedback loop.
The r(n) is defined as:

$$r_i(n) = \xi(|p_i(n)| - \sum_{j=1}^{q} x_{ij}(n)) \tag{13}$$

where q is the length of the input vector.
The function $\xi(.)$ is defined as:

$$\xi(v) = 1/2 + 1/2(erf(v/(2k))) \tag{14}$$

This function is implemented by the Ferroelectric Capacitor.

The equation of the first order recursive filter of the feedback loop is:

$$p_i(n) = ap_i(n-1) + g_i(n) \tag{15}$$

where $g_i(n)$ is the output dependent on $y_i(n)$ and $r_i(n)$

$$g_i(n) = \xi(r_i(n) - y_i(n)) \tag{16}$$

The state $mp_i(n)$ of the model neuron i can be rewritten as:

$$mp_i(n) = -\{v + p_i(n)r_i(n)\} + \sum_{j \in \tau}(\omega_{ij}(n)u_{fij}(n)) \tag{17}$$

The output $y_i(n)$:

$$y_i(n) = \xi(mp_i(n)) \tag{18}$$

The process of accumulation voltage and transformation to time is described by the block diagram named as membrane potential in Figure 4. From the figure we notice that $\theta_1(n)$ depends on the feedback of the output and on the input signals. The filter of the feedback loop is a first order recursive filter with the coeficient equal to a. The absolute value of the filter output $(pi(n))$ subtracts from the sum of the inputs. Until the filter output reaches the sum of the inputs, the feedback loop accumulates voltage. This is done, by passing the response of $|p_i(n) - \sum_{j=1}^{q} x_{ij}(n)|$ through the $\xi(v)$ function (Eq. 7), resulting in $r_i(n)$. The $\xi(v)$ function is implemented by hystheresis of the FeCapacitor. The result of $r_i(n)$ is equal to 1 when the output of the filter is equal or greater to the sum of the input signal, and 0 otherwise. The $r_i(n)$ is multiplied by $(p_i(n))$ resulting in the dynamic threshold that is the response of the membrane po- tential block diagram. The dynamic threshold accumulates charge until a spike occurs and the filters are reset. Our work has mapped the biological behavior of accumulating charge to create the receptor field by adding the membrane potential block diagram described in Figure 4 to the spiking neuron model. Using this approach it is possible to solve non-linear problems (XOR - problem) with a single neuron with the hard limiter function as activation function $(\xi(v))$.

5. The simulations of the models in simulink by Matlab

5.1 FePerceptron model

The results show the the simulation of the FeNeuron performing the logic gates, AND, OR, NAND and NOR. The model performs thresholding operations with a very simple architecture. It was developed in MATLAB environment.

The simulated gates have an architecture of a single neuron. The synaptic weights, ω_1 and ω_2 of the model were adapted iteration-by-iteration basis. For the adaptation process, the neuron was trained with the error-correction rule as a convergence algorithm. This algorithm uses a supervised paradigm which means that the desired response is presented in the training process. The desired response is the output of the logic gate that has been simulated with the respective input, following the well-known truth table of logic gates.

The algorithm can be described as follow:

Variables and Parameters:

The inputs: $\mathbf{x}(n) = [\theta \;\; x_1 \;\; x_2]^T$

The weights: $\mathbf{w}(n) = [\omega_0 \;\; \omega_1 \;\; \omega_2]^T$

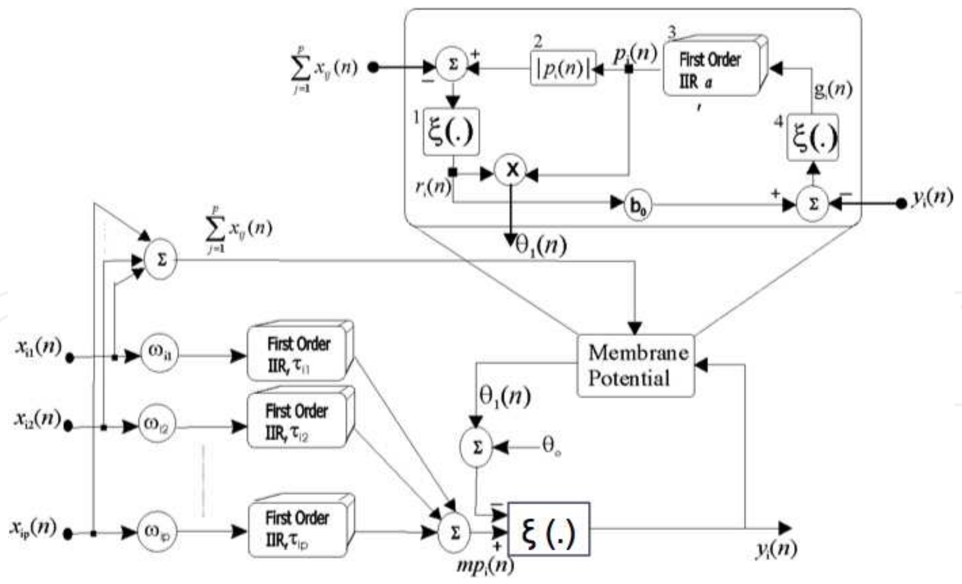


Fig. 4. The Discrete Spiking FeNeuron Model

The actual response: $y(n)$
The desired response: $d(n)$
The learning rate, or a positive constant less than a unit: η
The mean square error: $error(n)$

1. Initialization: Randomize $\mathbf{w}(n)$. Perform the computations from $n = 0, 1, 2, ...$ until the mean square error is minimum.
2. Computation of the actual response with (18).
3. Adaptation Process:

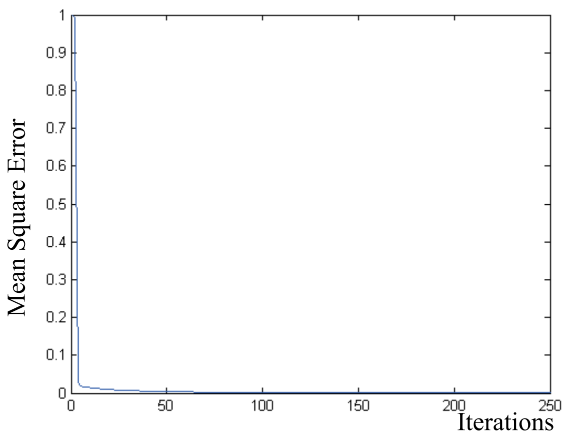
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[e(n)]\mathbf{x}(n) \tag{19}$$

$$e(n) = d(n) - y(n) \tag{20}$$

$$error(n) = \frac{1}{2}\sum_{j\in\Gamma} e^2(n) \tag{21}$$

   where Γ includes all neurons in the output layer of the network. In our case, only a single neuron is used.
4. Increment time step n by one and goes back to 2.

After the training process, the weights were computed and can be used to simulate the logic gates. The learning curves are shown in Figure 5 with the computed parameters.
The model performs thresholding operations with a very simple architecture. The Boolean functions performed by the model is soft programmable. This is accomplished by only adjusting the weight values of the synaptic connections. It is a very simple model that was easily implemented by software and can be extended to hardware implementations. As hardware implementations, this model brings the contribution of being very simple and can perform several functions with only changing the free parameters of the structure that can be soft-programmable.



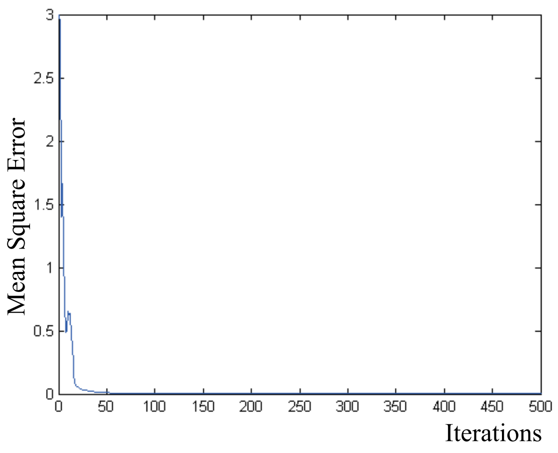
AND GATE

Weights = [-0.3759 0.6389 0.6389]

Actual Output = [-0.0006 0.0015 0.0015 0.9981]

Desired Output = [0 0 0 1]

Minimum error = 0.0000059874



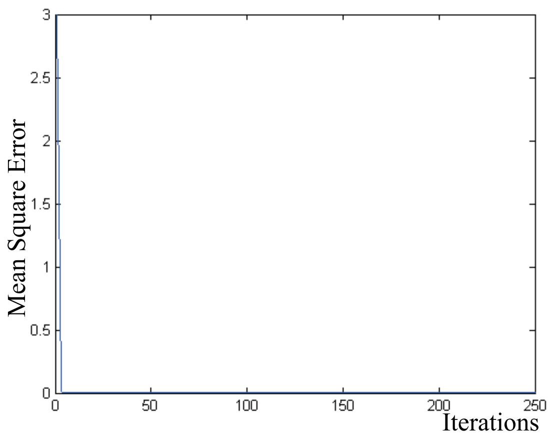
NAND GATE

Weights = [1.5709 -0.6503 -0.6503]

Actual Output = [1.0000 0.9988 0.9988 0.0017]

Desired Output = [1 1 1 0]

Minimum error = 0.0000059874



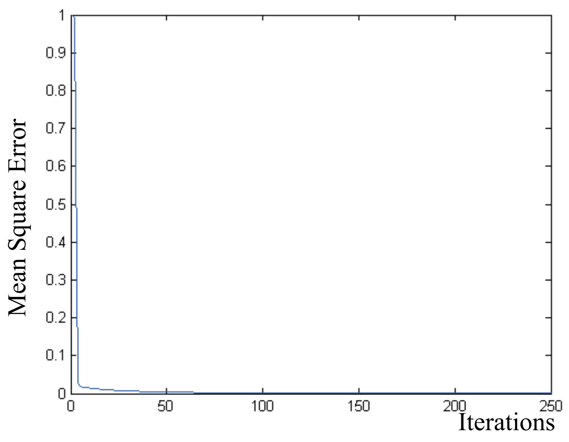
OR GATE

Weights = [0.2528 0.7089 0.7089]

Actual Output = [0.0013 0.9995 0.9995 1.0000]

Desired Output = [0 1 1 1]

Minimum error = 0.0000020539



NOR GATE

Weights = [0.9145 -0.6688 -0.6688]

Actual Output = [0.9986 0.0011 0.0011 -0.0009]

Desired Output = [1 0 0 0]

Minimum error = 0.0000052702

Fig. 5. The learning curves with the parameters for each computed logic gate. The inputs were (0 0), (0 1), (1 0) and (1 1). The threshold was considered equal to +1 for all gates.

5.2 Spiking FeNeuron model

In a digital simulation, the time period, n, is called a time slice. The four major steps of computing are:
1. Input phase: The input of each synaptic dendrite connection multiplied with the respective synaptic weight.

$$\alpha_{ij}(n) = \omega_{ij}(n)x_{ij}(n) \tag{22}$$

2. Filter phase: The signals from the input phase pass through the recursive filters.

$$ufij(n) = \alpha_{ij}(n) + e^{-\frac{T}{\tau_{ij}}} u_{fij}(n) \tag{23}$$

3. Output phase: The sum of the signals from the filter phase are summed to produce the membrane potential. If the membrane potential exceeds the dynamical threshold, an output spike is generated.
The state $mp_i(n)$ of the model neuron i can be rewritten as:

$$mp_i(n) = -\{v + p_i(n)r_i(n)\} + \sum_{j\in\tau}(\omega_{ij}(n)u_{fij}(n)) \tag{24}$$

$$p_i(n) = ap_i(n-1) + g_i(n) \tag{25}$$

$$g_i(n) = \xi(r_i(n) - y_i(n)) \tag{26}$$

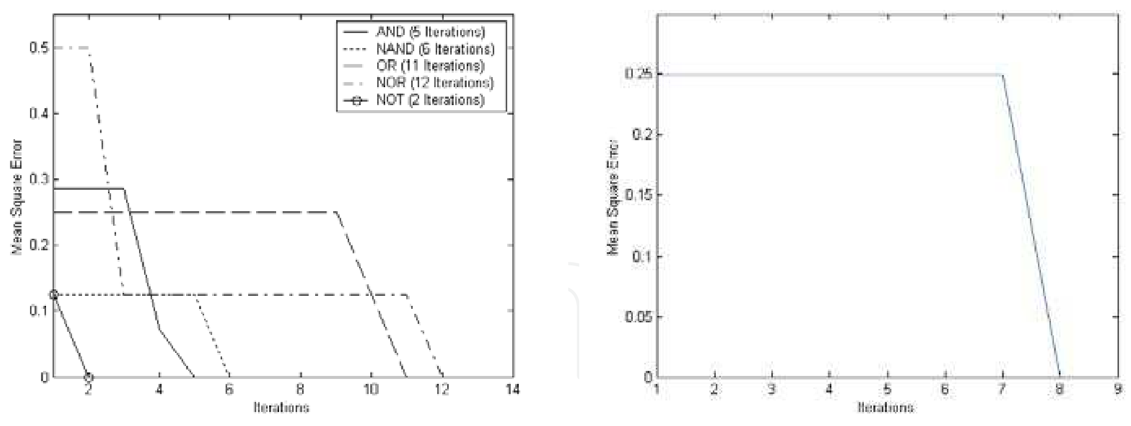
$$y_i(n) = \xi(mp_i(n)) \tag{27}$$

$$\xi(v) = \frac{1}{2} + \frac{1}{2}erf(\frac{v}{2k}) \tag{28}$$

If the spike is generated, the filters are reset. Otherwise the filter still accumu- lates potential.
4. Learning phase: The synaptic weights are adjusted with the Steepest Descent method.

$$\omega_{ij}(n+1) = \omega_{ij}(n) + \gamma(d_i(n) - y_i(n))x_{ij}(n) \tag{29}$$

where $d_i(n)$ is the desired output, $y_i(n)$ is the actual output, $x_{ij}(n)$ is the input, and γ is the learning rate.
These computation steps will be used in the logical functions problems. The computation of Boolean functions is a classification problem and as such it consists of separating the data into classes based on a discriminant function. There are two classes (0 or 1), and the input space is composed of four entries, each one with length two, except for the NOT logical gate that has one input and one output that negates the input.
Figure 6 (a) shows the learning curve for all logical gates and Figure 6 (b) shows for the XOR. The training phase is the same for all Boolean functions, we just have to modify the desired output to suit each logical gate. The weights and the filter coeficient are summarized in table I. The learning rate used was 0.01 and the filter coeficients of the first order recursive filter (τ_{ij}) were 0.01. Depending on the learning rate and the initial values of the weights the convergence rate changes.
Figure 7 shows the new symbol adopted for the neural logical gates and the output vector after the training process. The result is a perfect truth table of the respective logical gates. Table 1 shows the values for the decay constants, the weights and the learning rates for each logical



(a) The logical gates learning curves.

(b) The XOR logical gate learning curves.

Fig. 6. Learning curves.

gate. The Spiking FeNeuron model is composed by the input, $x_i(n)$, by the synaptic weights, $\omega_i(n)$, passing by the RC filters. The result is applied to the ferroelectric capacitor ($\xi(.)$) performing the output of the model. The phenomenon of the hysteresis loop is used to act as the the activation function. Using one side of the hysteresis that can be easily simulated as an error function. The simplicity of the Integrate-and Fire (IF) model is a good advantage. Others models, as quadratic IF, IF with adaptation, integrate- and-fire-or-burst and resonate-and-fire are extension and improvement of the integrate-and-fire model. These models are worried in capture the firing dynamics of real neurons (Janardan and Indranil, 2010). The main focus of this work is to generate a model that is able to compute and to be applied in engineering problems with a single neuron model and later with a network of neurons.

Parameters	AND	NAND	OR	NOT	XOR
Weights (ω_{ij})	0.48/0.48	-0.39/0.69	0.98/0.88	0/-1	0.568/-0.255
Decay constants (τ_{ij})	0.01/0.01	0.01/0.01	0.01/0.01	0.01/0.01	0.01/0.01
Learning rate (γ)	0.01	0.01	0.01	0.01	0.01

Table 1. Parameters of the Spiking FeNeuron.

5.3 Realization of the adaptive logic circuits

In this work the boolean logical gates are simulated by the Spiking FeNeuron showed in section 5.2. The logical gates are in turn used to construct flip-flop circuits and the flip-flop circuits are used to construct counters, shift-registers and adders. The logical gates, therefore, are used as the basic building blocks for all of the digital circuits and their purpose is to control the movement of binary data and instructions.

5.3.1 Design of the clock

All digital systems use a master timing signal called clock. This two state timing signal is usually generated from an analog source and may be digitally tuned to meet frequency and phase requirements. For the Spiking FeNeuron CPU, the clock was generated from an adapted XOR ring oscillator. A very simplified version of this oscillator is presented as the first stage of the frequency divider in 5. Here, the input A is tied to logic 1 thereby creating an inverter. The output of the inverter is then feedback to input B. The frequency of the oscillator will depend

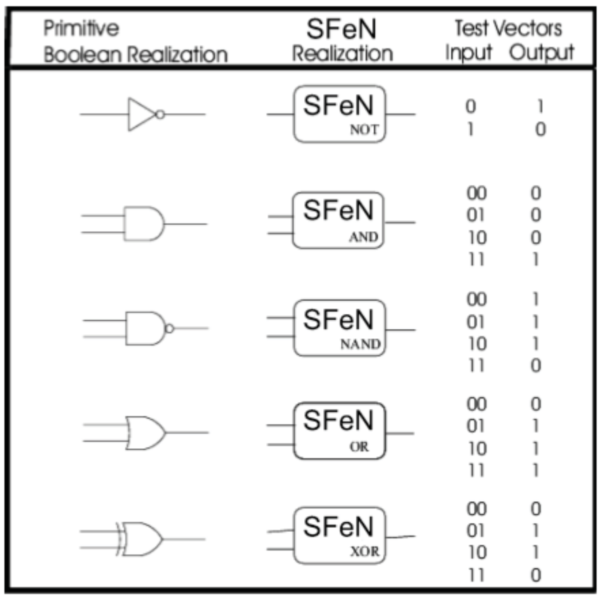


Fig. 7. The Spiking FeNeuron (SFeN) logical gates symbols and the test vectors after training.

on the delay of the feedback signal from the input to the output. Additional adapted XOR gates can be connected in scries to this one to some higher odd number to obtain the desired frequency of operation. As an observant reader may have noticed, a Spiking FeNeuron NOT gate could have been used in place of the Spiking FeNeuron XOR gate to derive the same results. The choice of the Spiking FeNeuron XOR gate does not provide any additional benefits over Spiking FeNeuron NOT gate but goes further to demonstrate the exibility of the Spiking FeNeuron logic. In standard CMOS logic, the NOT gate is almost always the only choice of a logic component for a ring oscillator due to its few transistor count of two. Compared to the 16-transistor count for a typical CMOS XOR gate, the area and ultimately cost savings in silicon makes the CMOS NOT gate the prime choice. In Spiking FeNeuron logic, however, each of the gates is derived from a single neuron trained to perform its function, thereby allowing tremendous area savings in hardware.

5.3.2 Design of the frequency divider circuit

A multi-stage frequency divider circuit can be implemented using addition and Spiking FeNeuron XOR gates. This circuit takes a clock as an input and provides three outputs with frequencies that are a divide by 2, by 4, and by 8 of the frequency of the input signal. The design schematic is presented together with the output waveform in Figure 8. On the schematic, CLK is the input clock signal, Y2 is the divide-by-2 output, Y3 is the divide-by-4 output, and Y4 is the divide-by-8 output.After training all of the gates, a function called Spiking FeNeuron was created which has two inputs: data vector and the option to select the desired logical gate required to perform a desired function. The output is the result from the selected gate.

5.3.3 Design of the D-type flip-flop

The D-type flip-flop is basically a SET-RESET latch with a small circuit modification. On the rising edge of the clock, the D input is latched to the output Q. The Spiking FeNeuron flip-flop logic circuit is shown in Figure 9. A test vector was generated to test the flip-flop in the example 1.

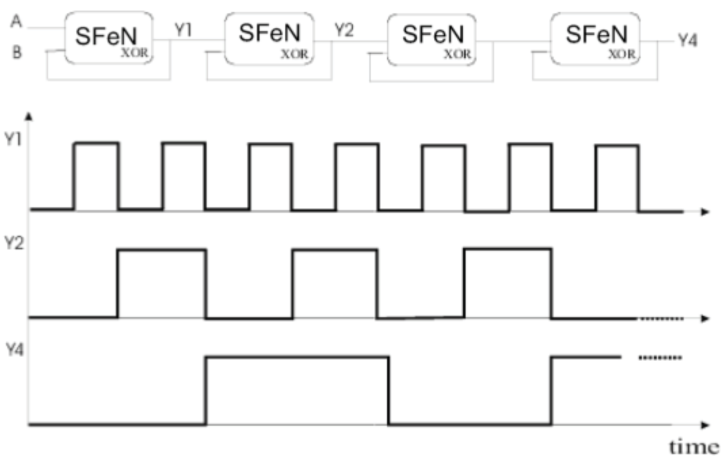


Fig. 8. The block diagram of the frequency divider with waveform.

MATLAB FUNCTION
function output = dff(data,clk)
data - input vector
clk - vector
output - response vector
EXAMPLE 1:
x = [0 0 1 1 1 0 1 0 1]
clk = [0 1 0 1 0 1 0 1 0]
output = dff(x,clk)
output = [0 0 0 1 0 0 0 0 0]

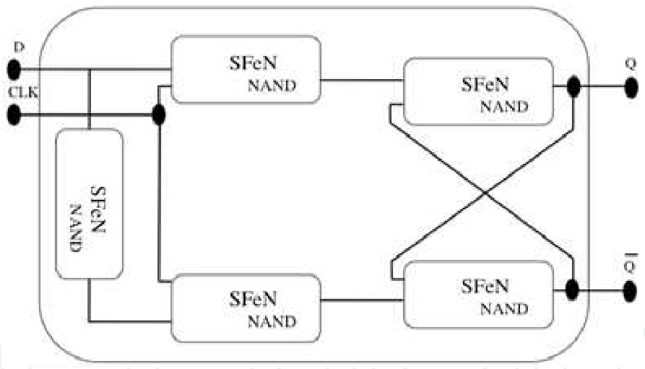


Fig. 9. The block diagram of D-flip-flop.

5.3.4 Design of the shift-register

A shift register is constructed using the flip-flop as shown in Figure 10. The shift register is a storage register that will move or shift the bits of the stored word either to the left or the right. The simulation of the Serial-In, Serial-Out (SISO) shift register is shown in example 3 with a test vector. The test vector with a 4-bit word [0110] is being applied to the shift registers input. The initial state of the shift register flip-flop output is 0. After the first clock pulse, the data stored is shifted one position to the right and the first bit of the applied serial word is shifted to the first position of the shifter register. After four clock pulses all the input data will be stored in the shift register. The summary of the test vector is shown in example 2.

MATLAB FUNCTION
function output = shiftreg(data)
data - input vector
clk - vector of the clock is generated inside the code
output - response vector
EXAMPLE 2:
output = shiftreg([1 0 0 1])
output = [0 0 0 0 1 0 0 0]

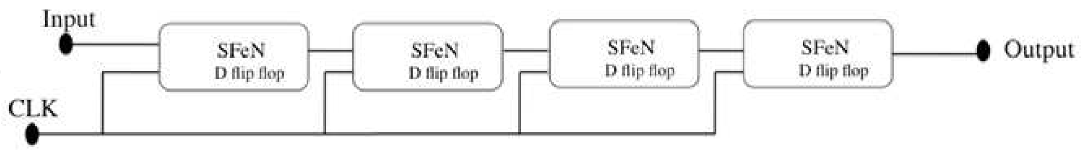


Fig. 10. The block diagram of the shift-register.

5.3.5 Design of the ALU

The ALU was construct using half-adder and full-adder circuits. The half-adder circuits were constructed using Spiking FeNeuron XOR and AND logic gates. The design schematic is presented on the right side of the Figure 11 where nodes A and B are the half-adder inputs, and S and carry denote the sum and the carry output signals respectively. The full-adder circuits were constructed from Spiking FeNeuron half-adders and Spiking FeNeuron logic gates. The design schematic is shown on the left side of the Figure 11 where A, B and CI represent the input and carry-in signals respectively. S and carry are the sum and carry outputs respectively. The full-adder was simulated for proper functionality. The results of this simulation are presented in example 3.
MATLAB FUNCTION
function [s,carryout] = fadder(data1,data2,carryin)
data1 - input vector
data2 - input vector
carryin - input of the carry
s - response vector of the sum
carryout - carry output

EXAMPLE 3:
data1 = [1 0 0 1]
data2 = [1 1 1 1]
$[s, c] = fadder(data1, data2, 0)$
s = [1 0 0 0]
c = 1

5.3.6 Design of a simple neural CPU

The Central Processing Unit contains an arithmetic-logic unit (ALU), a con- trol unit, and the registers for storage and manipulation of the data. The design of the CPU contains the ALU, a 32-bit 8x8 memory designed from Spiking FeNeuron D flip-flops. The system configuration of the CPU is shown on Figure 12. Information on the system bus which comprise CPU, memory control and data bhts was simulated with the use of switches. The binary instructions include memory and register access commands as well as ALU operational commands. The

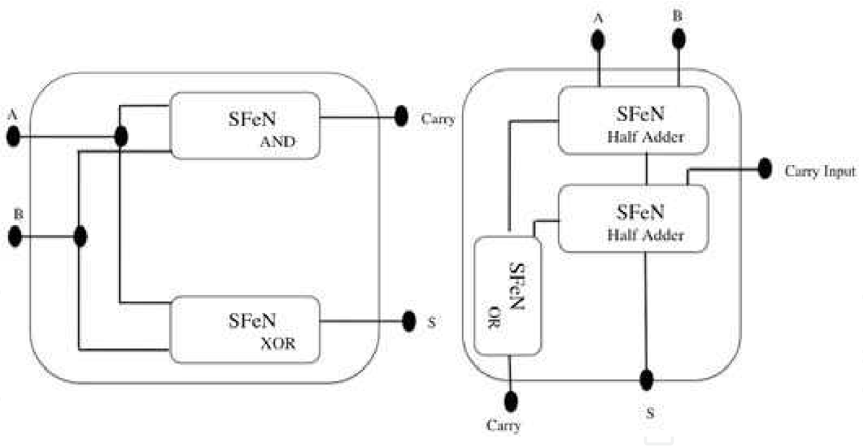


Fig. 11. The block diagram of the full adder.

microcode structure is shown on Table 2.
An example of some results for the instructions given to the CPU with 8 bits data is shown (Guerreiro et al., 2008).

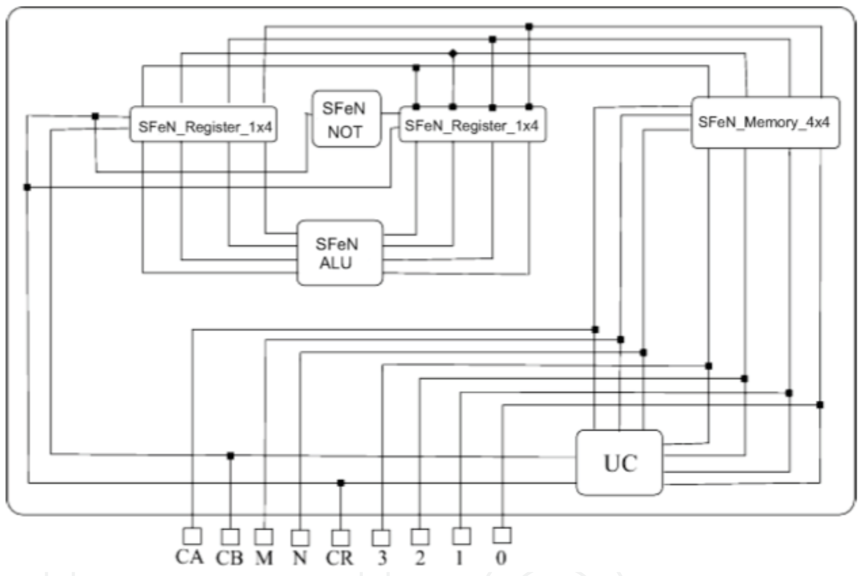


Fig. 12. The block diagram of the neural CPU.

Parameters	Name
CA	R/W operation memory
CB	R/W operation register
M and N	Memory Selector
CR	Register Selector
3/2/1/0	Data

Table 2. The microcode structure.

6. The Model of the FePerceptron in an FPGA

Now, this work is going to show the implementation of the FePerceptron model in a FPGA. For this implementation we are going to use the DSP builder tool of Altera Corporation. The DSP Builder technology allows you to go from system definition/simulation using the industry-standard the MathWorks/Simulink tools to the neuron implementation. The DSP Builder Signal Compiler block reads Simulink Model Files (.mdl) that are built using DSP Builder and MegaCoreÂő blocks and generates VHDL files and tool command language (Tcl) scripts for synthesis, hardware implementation, and simulation. The DSP builder automatically generate timing-optimized register transfer level (RTL) code based on high-level Simulink design descriptions (Altera, 2011).
In this way, first we developed the block diagram of the Simulink model of the FePerceptron which is shown in the Figure 13. The model is composed by the inputs, in this case two inputs as required by a Boolean logic gate, and the weights that were generated by the simulations in Matlab. After that the signal is summed and the output is generated passing the signal through the activation function that is implemented by the hystheresis of the FeCapacitor. The Figure 13 shows the implementation of the AND gate, for the other gates we only have to change the weights values, the same structure is used. The Table 3 shows the result of the simulations for the gates. Each gate is tested with the input vectors (Data1, Data2) and the output is seen by the display in Figure 13.

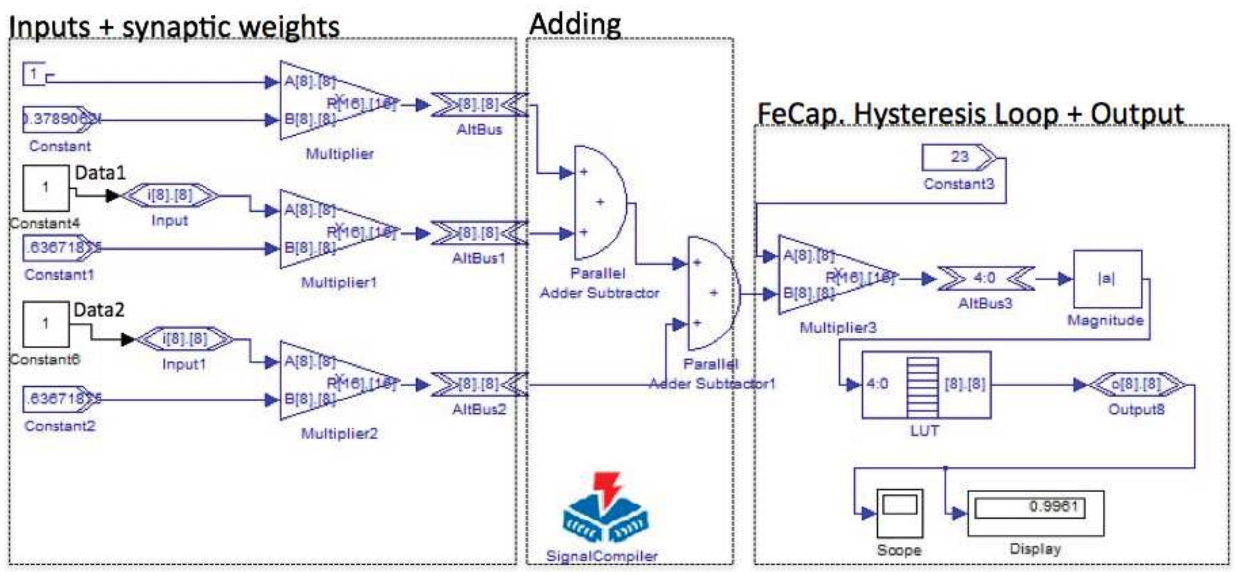


Fig. 13. The block diagram of the FePerceptron in Simulink (DSP Builder) for AND gate.

Data1	Data2	Display(AND)	Display(NAND)	Display(OR)	Display(NOR)
0	0	0	1	0	0.9961
0	1	0	0.9961	0.9973	0
1	0	0	0.9961	0.9973	0
1	1	0.9961	0	1	0

Table 3. The true table simulated by the simulink model of the FePerceptron.

The Simulink model then is converted to the RTL level code. Since the RTL level has a lot of details is not possible to show all in this work, more details can be found (VHDL, 2011).

7. Conclusion

The FeCapacitors have been embedded into LSIs as Ferroelectric Random Access Memory (FeRAM) and their reliability data have been accumulated for a long time. The capacitors are high impedance device, and it is an advantage for low power consumption, besides the configuration can be changed after packaging.
Thinking on this scenario, the FeCapacitor was choosen to be used in this work. It uses the phenomenon of the hysteresis loop of the FeCapacitor as the activation function for the artificial neuron models. We developed two models, the FePercetron and the FeSpiking Neuron Model, both models were first simulated in Matlab, and used to simulated the boolean functions. Since the FePerceptron were not able to simulated the XOR gate with a single neuron, because of the Perceptron characteristics. We were motivated to implement the FeSpiking that was based in the Extended Spiking Neuron Model and all logic gates were simulated, including the XOR. So, an adaptive simple CPU were developed, with simple logical circuits implemented, as registers, ALU, D-flip-flop as shown in section 4.
The FePerceptron and the FeSpiking Neuron Model presented the advantaged of being soft-programmable. This is accomplished by only adjusting the weight values of the synaptic connections without the need of changing all the architecture. It was firstly implemented by software verifying the success of the models.
From both models, first we chose the FePerceptron to be implemented in hardware because of the simplicity of the model. For this implementation we used the DSP builder tool of Altera Corporation. The DSP Builder Signal Compiler block read Simulink Model Files developed(.mdl) that were built using DSP Builder blocks and generated the VHDL files and the RTL level. This is the first step to develop more complex model as the FeSpiking Neuron Model, since the basic unit of the activation function (FeCapacitor) is already developed.
As hardware implementations, this model brings the contribution of being very simple, can save in silicon area, with low power consumption and being reconfigurable in two degrees of freedom, not only as characteristics intrinsic of the FPGA, but with the reconfigurability of the boolean gates. It is only necessary to change the values of the weights and the output is going to change to be the desired gate.

8. References

Altera Corporation. Diponível em: www.altera.com/products/dsp/dsp-builder.html. Acessado em: 10 de fevereiro de 2011.

Beiu, V.; Quintana, J. M.; Avendillo, M. J. (2003). *VLSI Implementations of Threshold Logic - A Comprehensive Survey*, Vol. 14, pp. 1217-1243.

Bermak, D.; Martinez, D. *A compact 3-D VLSI classifier using bagging threshold network ensembles* IEEE Transactions on Neural Networks 14(5) (2003) 1097âĂŞ1109.

Brown, B.; Yu, X.; Garverick, S. *A Mixed-mode analog VLSI continuous-time recurrent neural network* Proceedings of International Conference on Circuits, Signals and Systems, 2004,pp.104âĂŞ108.

Chen, Z.; Haykin, S; Becker, S. *Theory of monte carlo sampling-based alopex algorithms for neural networks*. Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 17-21.

Dias, Fernando Morgado; Antunes, Ana; Manuel Mota, Alexandre. *Artificial neural networks: a review of commercial hardware* Engineering Applications of Artificial Intelligence, v.17 n.8, p.945-952, December, 2004

Duong, T.A. *Cascade error projection: an efficient hardware learning algorithm* Proceedings of the IEEE International Conference on Neural Networks, vol. 1, Perth, Australia, 1995, pp. 175-178.

Duong, T.A.; Stubberud, A.R., Convergence analysis of cascade error projection: an efficient hardware learning algorithm. International Journal of Neural System. v10 i3. 199-210.

D'Acierno, A. *Back-propagation learning algorithm and parallel computers: the CLEPSYDRA mapping scheme*. Neurocomputing. v31. 67-85.

Fakhraie, S.M.; Farshbaf, H; Smith, K.C. *Scalable closed-boundary analog neural networks*. IEEE Transactions on Neural Networks. v15. 492-504.

Glesner, M.; Poechmueller, W. *Neurocomputers: An Overview of Neural Networks in VLSI*. 1994. Chapman and Hall, London.

Guerreiro, Ana Maria GuimarÃčes; McMillan, Larry; Araujo, Carlos A Paz de. *Adaptive Logic Synthesis by Ferroelectric Spiking Neuron Circuits*. Integrated Ferroelectrics, v. 100, p. 238-253, 2008.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd Edition.

Heemskerk, J. *Overview of neural hardware* Neurocomputers for Brain-Style Processing, Design, Implementation and Application, 1995.

Kumar, V.; Shekhar, S.; Amin, M. B. *A Scalable Parallel Formulation of the Backpropagation Algorithm for Hypercubes and Related Architectures* IEEE Transactions on Parallel and Distributed Systems, v.5 n.10, p.1073-1090, October 1994.

Ienne, Paolo; Cornu, Thierry; Kuhn, Gary. *Special-purpose digital hardware for neural networks: an architectural survey* Journal of VLSI Signal Processing Systems, v.13 n.1, p.5-25, Aug. 1996.

Jabri, Marwan; Flower, Barry. *Weight perturbation: An optimal architecture and learning technique for analog vlsi feedforward and recurrent multilayer networks.* Neural Computation, v.3 n.4, p.546-565, Winter 1991.

Janardan, Misra; Indranil, Saha. *Artificial neural networks in hardware: A survey of two decades of progress*. Journal Neurocomput. vol 74. 2010. ISSN: 0925-2312. Elsevier Science Publishers B. V.

Kung, S.Y. *Digital Neural Networks* Prentice-Hall, Upper Saddle River, NJ, USA, 1992.

Lehmann, Torsten; Bruun, Erik; Dietrich, Casper. *Mixed analog/digital matrix-vector multiplier for neural network synapses* Analog Integrated Circuits and Signal Processing, v.9 n.1, p.55-63, Jan. 1996.

Lamela, H.; and Ruiz-Llata, M. *Optoelectronic neural processor for smart vision applications*. Imaging Science Journal. v55 i4. 197-205.

Lenne, P. *Digital hardware architectures for neural networks* Speedup Journal 9(1)(1995)18âĂŞ25.

McCulloch, W. S.; Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity*, Bull. Math. Biophysiol., vol. 5, pp. 115-133.

Mead, C. *Analog VLSI and Neural Systems* Addison-Wesley,Boston,MA, USA, 1989.

Miller, S. L.; Schwank, J. R.; Nasby, R. D.; and Rodgers, M. S. (1991). *Modeling Ferroelectric capacitor switching with asymmetric nonperiodic input signals and arbitrary initial conditions*. J. Appl. Phys., vol. 70, no. 5, pp. 2949-2860.

Moerland, P.D.; Fiesler, E; and Saxena, I. *Incorporation of liquid-crystal light valve nonlinearities in optical multilayer neural networks*. Applied Optics. v35. 5301-5307.

Nedjah, Nadia; Mourelle, Luiza de Macedo. *Reconfigurable hardware for neural networks: binary versus stochastic* Neural Computing and Applications, v.16 n.3, p.249-255, May 2007.

Patnaik, L.M.; Rao, R.N. *Parallel implementation of neocognitron on star topology: theoretical and experimental evaluation*. Neurocomputing. v41. 109-124.

RÃąk, ÃĄdÃąm; SoÃşs, BalÃązs Gergely; Cserey, GyÃűrgy. *Stochastic bitstream-based CNN and its implementation on FPGA* International Journal of Circuit Theory and Applications, v.37 n.4, p.587-612, May 2009.

Rosemblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. Psych. Revue, vol. 65, pp. 386-408.

Sheikholeslami, A.; Gulak, P. Glenn (1997). *A Survey of Behavioral Modeling of Ferroelectric Capacitors*. IEEE Trans. on Ultrasonics, Ferroelectrics, and Frequency Control vol. 44, no. 4, pp. 917-923.

Schmid, A.; Leblebici, Y.; and Mlynek, D. *A mixed analog digital artificial neural network with on chip learning*. IEE Proceedings-Circuits, Devices and Systems. 2004. v146. 345

Schrauwen, B.; D'Haene, M. *Compact digital hardware implementations of spiking neural networks* J. Van Campenhout (Ed.), Sixth FirW Ph.D. Symposium, 2005, in CD.

Shibata, T.; Ohmi, T. (1991). *An intelligent MOS transistor featuring gate-level weighted sum and threshold operations.* Technical Digest of International Electron Devices Meeting, Washigton DC, pp. 919-922.

Smieja, F. J. *Neural network constructive algorithms: trading generalization for learning efficiency* Circuits, Systems, and Signal Processing, v.12 n.2, p.331-374, 1993.

Sundararajan, N.; Saratchandran, P. *Parallel Architectures for Artificial Neural Networks: Paradigms and Implementations* IEEE Computer Society Press, Los Alamitos, CA, 1998.

Strey, Alfred; Avellana, Narcis. *A New Concept for Parallel Neurocomputer Architectures* Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II, p.470-477, August 26-29, 1996.

Tokes, S.; OrzÃš, L.; Vr, G.; Roska, T. *Bacteriorhodopsin as an analog holographic memory for joint fourier implementation of CNN computers* Technical Report DNS-3-2000, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Hungary, 2000.

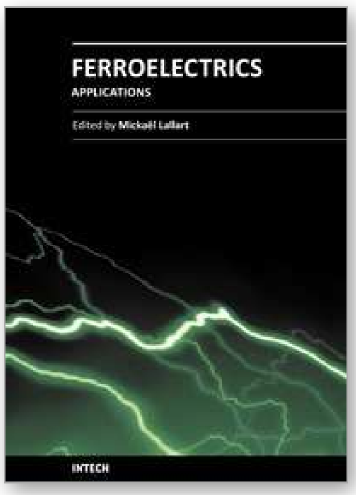
Valle, M. *Smart Adaptive Systems on Silicon*. 2005. Springer, Dordrecht, The Netherlands.

Verleysen, Michel; Thissen, Philippe; Voz, Jean-Luc; Madrenas, Jordi. *An Analog Processor Architecture for a Neural Network Classifier*, IEEE Micro, v.14 n.3, p.16-28, June 1994.

VHDL. Diponível em: ftp://alan:web@users.dca.ufrn.br. Acessado em: 20 de fevereiro de 2011.

Zhu, J.; Sutton, P. *FPGA implementations of neural networks-a survey of a decade of progress*. Field-Programmable Logic and Applications, vol. 2778. pp. 1062-1066.

Yang, F.; Paindavoine, M. *Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification*. IEEE Transaction on Neural Networks. v14 i5. 1162-1175.



Ferroelectrics - Applications
Edited by Dr. MickaÃ«l Lallart

ISBN 978-953-307-456-6
Hard cover, 250 pages
Publisher InTech
Published online 23, August, 2011
Published in print edition August, 2011

Ferroelectric materials have been and still are widely used in many applications, that have moved from sonar towards breakthrough technologies such as memories or optical devices. This book is a part of a four volume collection (covering material aspects, physical effects, characterization and modeling, and applications) and focuses on the application of ferroelectric devices to innovative systems. In particular, the use of these materials as varying capacitors, gyroscope, acoustics sensors and actuators, microgenerators and memory devices will be exposed, providing an up-to-date review of recent scientific findings and recent advances in the field of ferroelectric devices.