Artificial Neurons Based on CMOS β-Driven Threshold Elements with Functional Inputs[1]

V. VARSHAVSKY[1], V. MARAKHOVSKY[2], I. LEVIN[3]
[1,3] School of Engineering, Bar Ilan University, ISRAEL
[2] The University of Aizu, JAPAN

Abstract - This paper deals with a CMOS based artificial neuron implemented by threshold elements. We consider the artificial neuron as a threshold element with controlled inputs having weights formed during a learning process. A so-called β-driven threshold element is used for in the scheme of the neuron. Functioning of this element is described in a specific ratio form. The β-driven implementation is based on using summarized conductivities of n-and p-chains of a CMOS gate as the ratio of weighted sums. The threshold element has a wider functional capability in comparison with the traditional functional basis. Moreover, its functional capability can be enriched.

We propose a method for increasing the functional capability of the threshold element by introducing so-called functional inputs. Each functional input corresponds to a Boolean sum (or product) of a particular subset of input variables. This sum (or product) serves as a single input of the threshold element. It is shown that introducing functional inputs enables expansion of the functional capability of β-driven elements up to the capability to implement an arbitrary monotonic function. The CMOS based implementation of the β-driven threshold element with newly proposed functional inputs is presented. Methods of the current stabilization of functional inputs are proposed. In the proposed implementation of the artificial neuron, each input weight is determined by the current value via a suitable current stabilizer. This value can be effectively controlled by the value of the voltage at the gate of one of the current stabilizer's transistors.

The paper presents examples of the SPICE simulation of behavior of the proposed artificial neuron in the modes of learning and maintaining the input weights values.

Key-Words: Artificial neuron, threshold element, learnable circuit, CMOS circuits, SPICE-simulation.

1 Introduction

The great interest to threshold elements and threshold logics, lasting for tens of years [1-6], is caused, to our mind, first of all by wider functional threshold elements' capabilities in comparison with the traditionally based ones' (i.e. AND, NAND, OR, NOR etc.), and by the fact, that threshold elements may be used as a functional basis for artificial neural networks.

The effectiveness of using the threshold basis depends first of all on the implementation complexity of a threshold element.

In [7-12] we suggested a so-called β-driven CMOS threshold element that requires only one transistor per a functional input, having a weight that could be determined by the width of this transistor. It's hard to imagine the implementation to be simpler than this one.

The base for β-driven implementation was a fairly simple transformation of a regular analytic representation of the threshold function to a ratio form [7].

In the traditional representation of the threshold function

$$Y = Sign(\sum_{j=0}^{n-1} \omega_j x_j - \eta),\ Sign(A) = \begin{cases} 1 & if \quad A \geq 0 \\ 0 & if \quad A < 0 \end{cases}, \quad (1)$$

where ω_j - the weight of the j[th] input, η - threshold, we select some arbitrary S-subset of variables, for which: $\sum_{i \in S} \omega_i = \eta$.

Then:

$$\sum_{j=0}^{n-1} \omega_j x_j - \eta = \sum_{k \notin S} \omega_k x_k - (\eta - \sum_{i \in S} \omega_i x_i) = \quad (2)$$
$$= \sum_{k \notin S} \omega_k x_k - \sum_{i \in S} \omega_i (1 - x_i) = \sum_{k \notin S} \omega_k x_k - \sum_{i \in S} \omega_i \bar{x}_i$$

Hence:

$$Y = Sign(\sum_{j=0}^{n-1} \omega_j x_j - \eta) = Rt\left(\frac{\sum_{k \notin S} \omega_k x_k}{\sum_{i \in S} \omega_i \bar{x}_i} \right), \text{ where}$$

$$Rt(B) = \begin{cases} 1 & if\ B \geq 1 \\ 0 & if\ B < 1 \end{cases}. \quad (3)$$

[1] The project is financed by the Ministry of Industry and Trade of Israel, Magneton Agency (file N 27995).

To avoid uncertainty of 0/0 type, it's enough to shift the threshold in the initial determination of a threshold function by some δ, where $0 < \delta < 1$, as follows:

$$Y = Sign(\sum_{j=0}^{n-1} \omega_j x_j - \eta + \delta) = Rt\left(\frac{\delta + \sum_{k \notin S} \omega_k x_k}{\sum_{i \in S} \omega_i \bar{x}_i}\right),$$

$$\text{where } Rt(B) = \begin{cases} 1 & if\ B > 1 \\ 0 & if\ B < 1 \end{cases}. \qquad (4)$$

β-driven implementation, following from the ratio form, is based on changing the ratio of weight sums to the ratio of summarized conductivities of n- and p-chains of CMOS gate (Fig.1,a).

The functioning of the circuit in Fig.1,a is described in Table 1. In the table, every cell corresponding to one combination of input variable values is divided to 3 sub-cells. The upper left sub-cell contains the number of single conductivities of p-transistors, switched on, for the given values set of input variables. The upper right sub-cell contains the analogous number for n-transistors. The lower sub-cell contains the output function value (the inverter output).

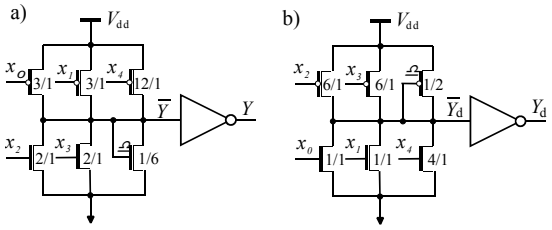

Figure 1. CMOS implementations of the threshold for ratio 1/3 for a certain function and its dual function.

Table 1. Functional table for the circuit shown in Fig.1,a.

$x_0x_1x_2$ x_3x_4	000		100		010		110		001		101		011		111	
00	6	δ	6	2+δ	6	2+δ	6	4+δ	2	δ	2	2+δ	2	2+δ	2	4+δ
	0		**0**		**0**		**0**		**0**		**1**		**1**		**1**	
10	5	δ	5	2+δ	5	2+δ	5	4+δ	1	δ	1	2+δ	1	2+δ	1	4+δ
	0		**0**		**0**		**0**		**0**		**1**		**1**		**1**	
01	5	δ	5	2+δ	5	2+δ	5	4+δ	1	δ	1	2+δ	1	2+δ	1	4+δ
	0		**0**		**0**		**0**		**0**		**1**		**1**		**1**	
11	4	δ	4	2+δ	4	2+δ	4	4+δ	0	δ	0	2+δ	0	2+δ	0	4+δ
	0		**0**		**0**		**1**		**1**		**1**		**1**		**1**	

Table 2, represents the function that is dual to the initial one.

Table 2. Functional table for the circuit shown in Fig.1,b.

$x_0x_1x_2x_3x_4$	000		100		010		110		001		101		011		111	
00	4+δ	0	2+δ	0	2+δ	0	δ	0	4+δ	4	2+δ	4	2+δ	4	δ	4
	0		**0**		**0**		**0**		**0**		**1**		**1**		**1**	
10	4+δ	1	2+δ	1	2+δ	1	δ	1	4+δ	5	2+δ	5	2+δ	5	δ	5
	0		**0**		**0**		**1**		**1**		**1**		**1**		**1**	
01	4+δ	1	2+δ	1	2+δ	1	δ	1	4+δ	5	2+δ	5	2+δ	5	δ	5
	0		**0**		**0**		**1**		**1**		**1**		**1**		**1**	
11	4+δ	2	2+δ	2	2+δ	2	δ	2	4+δ	6	2+δ	6	2+δ	6	δ	6
	0		**0**		**0**		**1**		**1**		**1**		**1**		**1**	

Indeed, the function presented in Table 2 is:

$$Y_1 = x_0x_2x_3 \vee x_1x_2x_3 \vee x_0x_4 \vee x_1x_4 \vee x_2x_4 \vee x_3x_4 = Sign(x_0 + x_1 + 2x_2 + 2_3 + 4x_4 - 5). \qquad (5)$$

It follows from the definition of the dual function and (6) that

$$Y_d = \overline{\bar{x}_0\bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_0\bar{x}_1\bar{x}_4 \vee \bar{x}_2\bar{x}_4 \vee \bar{x}_3\bar{x}_4} = (x_0 \vee x_1 \vee x_2 \vee x_3)(x_0 \vee x_1 \vee x_4)(x_2 \vee x_4)(x_3 \vee x_4) = x_0x_2x_3 \vee x_1x_2x_3 \vee x_0x_4 \vee x_1x_4 \vee x_2x_4 \vee x_3x_4 = Y_1. \qquad (6)$$

The dual function in the threshold basis keeps input weights, and the threshold is calculated as the following:

$$[Sign(\sum_{j=0}^{n-1} \omega_j x_j - \eta)]_d = Sign(\sum_{j=0}^{n-1} \omega_j x_j - \sum_{j=0}^{n-1} \omega_j + \eta - 1) \qquad (7)$$

The most attractive feature of threshold functions is their functional plasticity, i.e. the possibility of changing the function, realized by threshold element, by means of simple changing input and output weights. This feature becomes important only if the input and the output weights control may be realized in a simple way.

So long as transistor conductivity depends monotonously on its gate voltage, the simplicity of realizing the control of the output weight in β-driven CMOS threshold elements becomes obvious [9,10] (Fig.2).

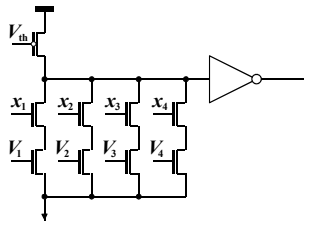

Figure 2. Implementation of the threshold function $Y = Sign[\sum_{j=1}^{4} \varphi_n(V_j)x_j - \varphi_p(V_{th})]$.

A threshold element realizes the threshold function, which is monotonic one. However, there are monotonic functions that are not threshold. Among monotonic functions of four (and more) variables there are some, which are not threshold, for example: $y = x_0x_1 + x_2x_3$.

As it can be seen, this function describes the behavior of a two-input multiplexer – the circuit, which is significant for a plenty of implementations.

In this paper we propose methods enabling to expend functional capabilities of β-driven elements right up to the capability to implement an arbitrary monotonic function.

We describe an artificial neuron based on the proposed threshold element having functional

inputs. We consider the whole neuron being a threshold element with controlled inputs having weights formed during a learning process. We propose CMOS implementations of both the β-driven threshold element, and the artificial neuron based on this element. We investigate characteristics of the proposed hardware implementations.

2 Threshold elements with functional inputs

First of all let us consider a number of circuits and functional examples.

The circuit, shown in Fig.3 a, realizes the second order symmetrical monotonic function

$$Y_1 = x_0x_1 \vee x_0x_2 \vee x_0x_3 \vee x_0x_4 \vee x_1x_2 \vee \vee x_1x_3 \vee x_1x_4 \vee x_2x_3 \vee x_2x_4 \vee x_3x_4 = \quad (8)$$
$$= Sign(x_0 + x_1 + x_2 + x_3 + x_4 - 2)$$

Let us split the n-channel part of the circuit to two channels and incorporate a current stabilizer, thus fixing the current equal to i0 in the channels (Fig. 3b). The function (9) will evidently change, as follows:

$$Y_2 = Sign(x_0 + z_1 + z_2 - 2) = \quad (9)$$
$$= x_0z_1 \vee x_0z_2 \vee z_1z_2$$

where $z_1 = x_1 \vee x_2$ and $z_2 = x_3 \vee x_4$, hence:

$$Y_2 = x_0x_1 \vee x_0x_2 \vee x_0x_3 \vee x_0x_4 \vee \vee x_1x_3 \vee x_1x_4 \vee x_2x_3 \vee x_2x_4 \quad (10)$$

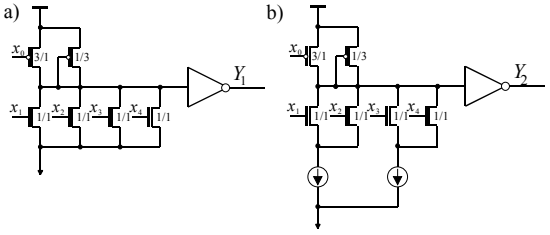

Figure 3. Implementations of functions: a) for the function (8); b) for the function (9).

It can be seen from the above example, that, if a particular subset of n-transistors controlled by $x_j \in R_k$ variables, is selected, and the total current is limited by unit value i_0 of a current stabilizer via this subset, the whole of this variable subset will operate in the output function as a single variable

$$z_k = \bigvee_{x_j \in R_k} x_j \ . \quad (11)$$

On the other hand, if a particular subset of p-transistors controlled by $x_i \in R_m$ variables, is selected, and the total current via this subset of transistors is limited by unit value i_0 of a current stabilizer, the whole of this variable subset of these variables operates in the output function as a single variable

$$z_m = \overline{\bigvee_{x_i \in R_m} \bar{x}_i} = \&_{x_i \in R_m} x_i \ . \quad (12)$$

For CMOS β-driven threshold elements define as a functional input the subcircuit (Fig. 4) consisting of:

a) transistors, connected in parallel and controlled by the variables of a particular subset,

b) a current stabilizer, connected in series, that the current via this subcircuit doesn't depend on the number of open transistors.

The positive feature of the proposed circuit is a fact that its implementability is independent on the rank of the product (sum) but on the number of products in the corresponding minimum form only.

We propose to use the threshold current stabilizer (Fig. 4) for generating (limiting) unit-operating current and hence, the higher degree of stabilization would be provided.

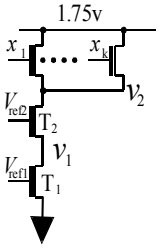

Figure 4. Current stabilizer: circuit of the functional input, having a current stabilizer on the couple of transistors connected in series;

It seems that any MOS-transistor provides current stability in saturated mode. Really, the current via T_1-transistor (Fig.4), under the condition $v_1 \geq V_{ref1} - V_{th}$ (saturated mode) doesn't depend on v_1, according to the first order Shockley equations $I_1 = \frac{\beta}{2}(V_{ref1} - V_{th})^2$.

However, taking channel length modulation effect into consideration, the linear dependence of the current via transistor on the voltage drop on it will be obtained as follows:

$$I_1 = \frac{\beta}{2}(V_{ref1} - V_{th})^2(1 + \lambda V_{ds}) = \frac{\beta}{2}(V_{ref1} - V_{th})^2(1 + \lambda v_1).$$

Results of the SPICE simulation for the proposed stabilizer are presented in Fig. 5.

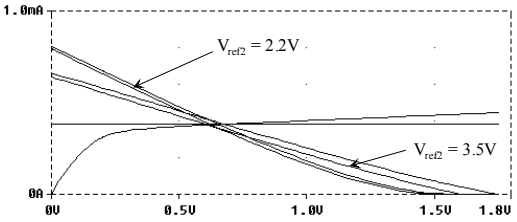

Figure 5. Behavior of the current stabilizer (Vref2=3.5V and Vref2=2.2V).

3. Learnable neuron circuit and the way of teaching

Implementing an artificial neuron as a digital-analog device has a number of advantages over software implementation. First, this is a sharp increase in performance. However, such an implementation, due to its internal analog nature, has rigid restrictions on the class of threshold functions that can be realized. These restrictions considerably decrease the functional possibilities of neural networks with fixed number of neurons.

In [9-12] we suggested a CMOS learnable neuron based on β-driven threshold element that consisting of synapses, β-comparator and output amplifier. It requires 5 transistors and one capacitor per a learnable synapse. The neuron has one remarkable property: its implementability depends only on the threshold value and does not depend on the number of logical inputs and their weights. This fact and low complexity make this artificial neuron fairly attractive for usage.

In our later works [13-18] the suggested neuron was modified by increasing the steepness of the β-comparator and incorporating two extra amplifiers with different thresholds. The circuit implementing such a neuron is given in Fig. 6.

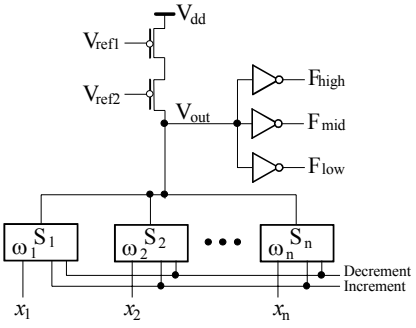

Figure 6. Learnable neuron circuit.

This circuit has two *p*-channel transistors with reference voltages V_{ref1} and V_{ref2} on their gates, which provide very sharp steepness of the β-comparator characteristic in the working point; *n* synopses S_j with weighted binary inputs x_j (adjusted weight of x_j is ω_j) and three output amplifier with different thresholds.

Synapse circuit with a binary input is represented in Fig. 7.

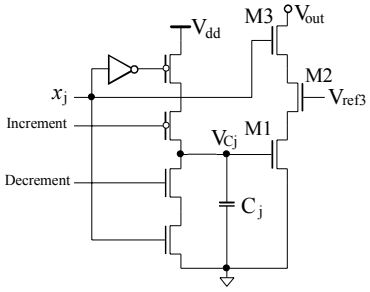

Figure 7. Synapse circuit with binary input.

We incorporate a current stabilizer into this circuit. The synapse is activated when $x_j = 1$. In this case transistor M3 is opened. Transistor M1 sets the value of the synapse current by means of control voltage V_{C_j} on its gate.

Circuits used for forming the control voltages that determine the weights of the input variables of a neuron just slightly depend on the way of the synapse implementation. Some of these circuits were studied (for example, in [19]) and they are of about the same structure. The difference between them is mainly associated with the memory element they use (capacitor or transistor with a floating gate) and with the way of representing the values of the input binary variables ({0,1} or {-1, +1}).

In our case, the voltage that controls the synapse current (i.e. variable weight) accumulates on the capacitor during the teaching. The capacitor charge is allowed to change only when the synapse is active, i.e. when the input variable equals "Log.1"'. The capacitor charge increase or decrease is realized by approximately the same quanta that determine the learning step. The learning step is specified on the base of the required accuracy of setting the control voltages. Its value can be controlled by choosing the amplitude and duration of "increment" and "decrement" signals.

What is the meaning of adding two extra output amplifiers in Fig. 6? One of the important parameters of a threshold element is the smallest (for the given input combinations) deviation of the β-comparator output voltage from the output amplifier threshold ($\pm\Delta V_{out}$). Actually, ΔV_{out} determines the maximum complexity of a

function that can be realized on a single neuron. In hyper-geometrical representation, a threshold element corresponds to a hyperflat separating the set of all input combinations to two subsets *T* and *F*. It is easy to understand that ΔV_{out} is determined by the shortest distance from *T* and *F* sets to the hyperflat. The same sets *T* and *F* can be separated by different hyperflats. Since in our learnable neuron the input weights change continuously, then the set of separating hyperflats also fills some subspace, and among this set of hyperflats there is a hyperflat with the biggest distance from *T* and *F*. Indeed, let us consider a threshold function

$y = x_1x_3 \vee x_2x_3 \vee x_4 = Sign(x_1 + x_2 + 2x_3 + 3x_4 - 3).$

The combinations closest to the separating hyperflat are T_0={1010,0110,0001} and F_0={1100, 0010}. The optimal representation of this threshold function in reduced ratio form is $y = Rt(0.4x_1 + 0.4x_2 + 0.8x_3 + 1.2x_4)$ with minimum residual (deviation of weighted sum of variables from the threshold) determining ΔV_{out} equal to ±0.2. In the same time, there is a correct implementation $y = Rt(0.22x_1 + 0.76x_2 + 0.8x_3 + 1.2x_4)$ with minimum residual equal to ±0.02.

The two extra amplifiers optimize ΔV_{out} during the learning. The amplifier thresholds F_{high} and F_{low} are selected so that their difference with the threshold of the main output amplifier F_{mid} is equal to the guaranteed ΔV_{out}. The neuron is taught so that the outputs of all the three amplifiers have the same value, providing the guaranteed value of ΔV_{out}.

If the input weight memory is implemented on a capacitor, then the threshold function degrades because of the capacitor charge leakage via parasite resistances. In this case, the extra amplifiers provide on-line recovery of synaptic weights values. Since a capacitor discharges fairly slowly, the wrong values of neuron output first appear at the outputs of the amplifiers F_{high} and F_{low} while the main output F_{mid} functions correctly. Thus, different values of F_{high} or F_{low} as compared to F_{mid} is a signal of some change in the synaptic weights. To correct them, we can use the same mechanism of forming increment and decrement signals as used in the learning.

The general structural scheme used when simulating the process of teaching the neuron to the given threshold logical function is shown in Fig. 8.

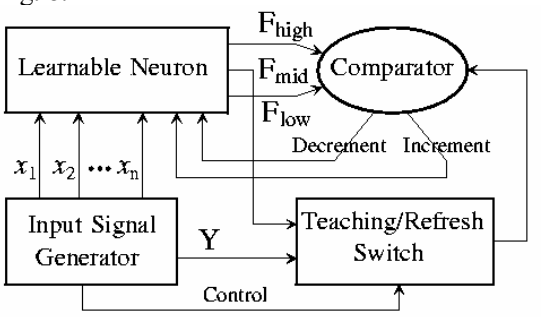

Figure 8. General scheme of the experiment

The generator of the input signals periodically produces sequences of value combinations of input variables $x_1, x_2, ..., x_n$ and the sequence of values that the given logical function *Y* takes on these combinations. Teaching/refresh switch passes to its output either the signal *Y* (when teaching) or the output signal F_{mid} (when refreshing). The comparator produces the signals "decrement" and "increment". The passive values of these signals are equal to "0" and "1" respectively. Their logical description looks as follows:

$Decrement = \overline{Y}F_{high}$ and

$Increment = \overline{Y} \vee F_{low}$ when teaching; and

$Decrement = \overline{F}_{mid}F_{high}$ and

$Increment = \overline{F}_{mid} \vee F_{low}$ when refreshing.

Physically, these signals are realized with limited amplitude and duration, determining the learning step.

In all the experiments with learnable neurons, there is an acute problem of selecting the threshold function for teaching what determines the simulation time. The experiment duration is often measured in hours and even days. A threshold function should satisfy the following requirements:

– short sequence of variable combinations checking all possible switches of the function value,

– covering a wide range of weights,

– high value of the threshold for the given number of variables.

Threshold functions that can be represented by Horner's scheme $x_n(x_{n-1} \vee x_{n-2}(x_{n-3} \vee x_{n-4}(...)))$ meet these requirements. For such functions, the sequence of integer values of the variable weights and the threshold with minimum sum forms the Fibonacci sequence. The length of the checking

sequence is n+1 for the Horner's function of n variables.

5 Concluding Remarks

In this paper we described a CMOS implementation of an artificial neuron based on a β-driven threshold element proposed by the authors. Using such an element as a functional basis of artificial neurons has a number of advantages in comparison with the traditional functional basis. In this paper we have focused on extension of functional capability of the threshold element.

The main contribution of the paper can be summarized as follows:

1. Introducing functional inputs into the β-driven threshold element to allow to increasing its functional capability up to the capability of realizing every monotonic Boolean function.
2. Implementation of the β-driven threshold element, having the proposed functional inputs, by a CMOS circuit, and investigation of its implementability.
3. Developing a method for current stabilization of the functional inputs.
4. Implementation of an artificial neuron using of the proposed threshold element, wherein the neuron has controlling inputs, weights of which are formed during the learning process, and wherein the current via a current stabilizer of a suitable input determines the weight of this input.

All the proposed solutions were simulated. The SPICE simulation results demonstrate the high efficiency of the solutions and consequently motivate further investigations of the artificial neurons based on β-driven elements.

References:

[1] S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Imminent in Nervous Activity", *Bulletin of Mathematical Biophysics*, 5, 1943, pp.115-133.

[2] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.

[3] T. Shibata, T. Ohmi, "Neuron MOS Binary-logic Integrated Circuits: Part 1, Design Fundamentals and Soft-hardware Logic Circuit Implementation", *IEEE Trans. Electron Devices*, Vol.40, No.5, 1993, pp. 974-979.

[4] T. Ohmi, T. Shibata, K. Kotani, "Four-Terminal Device Concept for Intelligence Soft Computing on Silicon Integrated Circuits". *Proc. of IIZUKA'96*, 1996, pp. 49-59.

[5] S.M. Fakhraie, K.C. Smith, *VLSI-Compatible Implementations for Artificial Neural Networks*. Kluwer, Boston-Dordrecht-London, 1997.

[6] Montalvo, R. Gyurcsik and J. Paulos, "Toward a General-Purpose Analog VLSI Neural Network with On-Chip Learning", *IEEE Transactions on Neural Networks*, Vol.8, No.2, March 1997, pp.413-423.

[7] V. Varshavsky, "Beta-Driven Threshold Elements", *Proceedings of the 8-th Great Lakes Symposium on VLSI*, IEEE Computer Society, Feb. 19-21, 1998, pp.52-58.

[8] V. Varshavsky, "Threshold Element and a Design Method for Elements", filed to Japan's Patent Office, Jan.30, 1998, the application number is JPA H10-54079.

[9] V. Varshavsky, "Simple CMOS Learnable Threshold Element", *International ICSC/IFAC Symposium on Neural Computation*, Vienna, Austria, Sept.23-25, 1998.

[10] V. Varshavsky, "CMOS Artificial Neuron on the Base of Beta-Driven Threshold Element", *IEEE International Conference on Systems, Man and Cybernetics*, San Diego, CA, October 11-14, 1998, pp.1857-1861.

[11] V. Varshavsky, "Synapse, Threshold Circuit and Neuron Circuit", filed to Japan's Patent Office on Aug. 7,1998, the application number is JPA-H10-224994.

[12] V. Varshavsky, "Threshold Element", filed to Japan's Patent Office on Aug. 12, 1998, the application number is JPA-H10-228398.

[13] V. Varshavsky and V. Marakhovsky, "Beta-CMOS implementation of artificial neuron", SPIE's 13th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls. Applications and Science of Computational Intelligence II, Orlando, Florida, April 5-8, 1999, pp.210-221.

[14] V. Varshavsky and V. Marakhovsky, "Beta-CMOS Artificial Neuron and Implementability Limits", Lecture Notes in Computer Science 1607, Engineering Applications of Bio-Inspired Artificial Neural Networks.

[15] Jose Mira, Juan V. Sanchez-Andves (Eds.). Proceedings of International Work-Conference on Artificial and Natural Neural Networks (IWANN'99)}, Spain, June 2--4, Springer, Vol.11, 1999, pp. 117-128.

[16] V. Varshavsky and V. Marakhovsky, "The Simple Neuron CMOS Implementation Learnable to Logical Threshold Functions", *Proceedings of International Workshop on Soft*

Computing in Industry'99 (IWSCI'99), June 16-18, Muroran, Hokkaido, Japan, IEEE, 1999, pp. 463-468.

[17] V. Varshavsky and V. Marakhovsky, "Implementability Restrictions on the Beta-CMOS Artificial Neuron", *Proceedings of the 6th International Conference on Electronics, Circuits and Systems (ICECS'99)*, September 5-8, 1999, Pafos, Cyprus, IEEE, 1999, pp. 401-405.

[18] V. Varshavsky and V. Marakhovsky, "Learning Experiments with CMOS Artificial Neuron", *Lecture Notes in Computer Science 1625, Computational Intelligence Theory and Applications, ed. by Bernard Reusch. Proceedings of the 6th Fuzzy Days International Conference on Computational Intelligence*, Dortmund, Germany, May 25-27, Springer, 1999, pp. 706-707.

[19] A. Montalvo, R. Gyurcsik, and J. Paulos, "Toward a General-Purpose Analog VLSI Neural Network with On-chip Learning", *IEEE Transactions on Neural Networks*, Vol.8, No.2, March 1997, pp. 413-423.