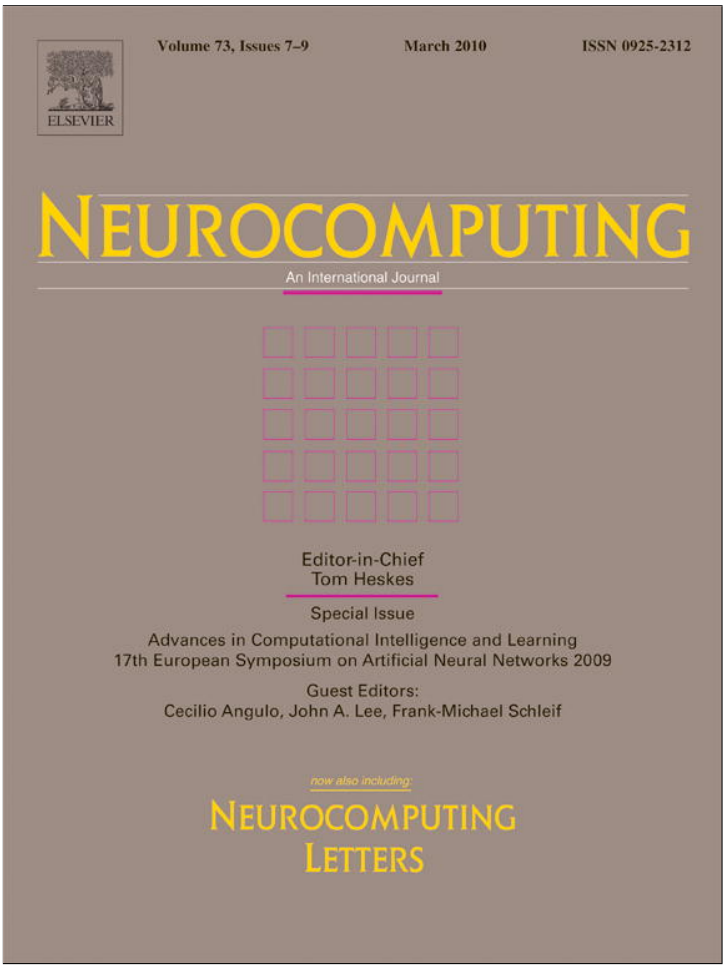

Neurocomputing 73 (2010) 1222–1234

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Łukasiewicz fuzzy logic networks and their ultra low power hardware implementation

Rafał Długosz [a,*], Witold Pedrycz [b,c]

[a] Swiss Federal Institute of Technology in Lausanne, Institute of Microtechnology, Rue A.-L. Breguet 2, CH-2000, Neuchâtel, Switzerland
[b] University of Alberta, Department of Electrical and Computer Engineering ECERF Building, Edmonton, Canada T6G 2V4
[c] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland

ARTICLE INFO

Available online 18 January 2010

Keywords:
Neural networks
Logic neurons
Fuzzy sets
CMOS analog devices
Ultra low power electronics

ABSTRACT

In this paper, we propose a new category of current-mode Łukasiewicz OR and AND logic neurons and ensuing logic networks along with their ultra-low power realization. The introduced circuits can operate in a wide range of the input signals varying in-between 10 nA and 10 μA. For low current values the operating point of transistors is set in the under threshold region. In this region, the mismatch between transistors exhibits a far stronger impact on the current mirror precision than the one observed in case of the strong inversion region. The proposed design alleviates this problem by reducing the number of current mirrors between the input and the output of the neuron and of the overall network to only one. Łukasiewicz operators require only summation and subtraction operations, which make them suitable for realization in analog current-mode technique. In this case even large number of input signals can be summed in a simple junction in a single step. This is the reason of choosing Łukasiewicz operations in the proposed circuit. Using other t-norm and t-conorm operations with multiplication and division operations would make the realization of the circuit very difficult and inefficient.

1. Introduction

Neurocomputing has emerged as a synonym of adaptive processing and significant learning capabilities. Fuzzy sets, treated as one of the branches of Granular Computing, are about processing information granules which are commonly encountered in human perception and information processing. The limitations of these two technologies (viz. fuzzy sets and neurocomputing) when being considered individually, are significantly eliminated by forming hybrid architectures known as neurofuzzy systems. In this synergistic environment we take full advantage of the learning capabilities of neural networks while benefiting from transparency of processing coming hand in hand with fuzzy sets. Fuzzy neural networks as introduced e.g., in [9,11] are examples of such hybrid architectures of neurocomputing.

In a nutshell, we can view these networks as an essential generalization of two-valued digital circuits which are now augmented by significant learning capabilities and flexibility to deal with continuous inputs viewed as truth values encountered in fuzzy logic and multivalued logic. As such, we can refer to a plethora of areas in which the use of neurofuzzy structures of this nature can be considered including: (a) pattern recognition, (b) rule-based systems (and approximate reasoning, in general), (c) logic-oriented models, (d) fuzzy and neural control, to name a few general categories.

Surprisingly enough, there has been a very limited research in hardware implementation and ensuing design practice of architectures of neurofuzzy systems. We can refer here to a few publications dealing with this subject matter, cf. [1,4,7,8]. In some sense, this is quite impeding considering a wealth of new potential applications of intelligent systems associated with mobile and pervasive computing as being vigorously promoted in the realm of ambient intelligence (AmI). One of the fundamental quests comes with regard to power consumption whose minimal level becomes highly beneficial to mobile devices. This somewhat implies intensive studies along the line of analog electronics.

The objective of this study is to introduce one of the variants of neurocomputing coming in the form of fuzzy neural networks whose neurons are realized by means of Łukasiewicz operators (*and* and *or* operators). The Łukasiewicz operators are a sound alternative which offers enough flexibility with regard to underlying logic processing which comes with appealing features supporting the hardware realization of these neurons.

The study is organized as follows. In Section 2, we start, with a brief introduction to Łukasiewicz logic operators. The motivation of choosing the Łukasiewicz operators in the context of realization

* Corresponding author.
E-mail address: rafal.dlugosz@epfl.ch (R. Długosz).

0925-2312/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2009.11.027

of logic processors (LP) has been highlighted in Section 3. In Section 4, we propose Łukasiewicz networks when using the "classic" approach to the realization of the fuzzy operations proposed by Yamakawa in [15]. The discussed limitations of this classic approach led us to the proposal of the new approach to the implementation of such networks (Section 5), along with simulation results reported at the transistor level. The conclusions are offered in Section 6. Throughout the paper we will adhere to the standard notation used in fuzzy sets.

2. Łukasiewicz logic operations

Łukasiewicz logic connectives arise as one of the possible realizations of logic operators applied to fuzzy sets [6]. Those are interesting examples of a broad category of logic connectives known as t-norms and t-conorms. Interestingly, those operations were originally introduced by Łukasiewicz in his original studies on three-valued and multiple-valued logic. More formally, the Łukasiewicz *and* operator is described as follows:

$$atb = \max(0, a+b-1) \tag{1}$$

while the *or* operator is governed by the following expression

$$asb = \min(1, a+b) \tag{2}$$

where the (logic) truth values a and b assume values in the unit interval.

These logic connectives can be arranged together in the constructs of *logic* neurons that are viewed as generic processing units encountered in logic neurocomputing. Logic neurons [10] offer functionality which embraces advantages of neural networks as far as learning is concerned and fuzzy logic which becomes advantageous with respect to the associated interpretation abilities. The logic neurons come with well-defined semantics, which is drawn from the nature of the underlying logic processing inherent to fuzzy computing. We encounter two fundamental categories of fuzzy neurons.

OR neuron: this neuron realizes an *and* logic aggregation of inputs $x=[x_1\ \ x_2 \ldots x_n]^{\mathrm{T}}$ with the corresponding connections (weights) $w=[w_1\ \ w_2 \ldots w_n]^{\mathrm{T}}$ and then summarizes the partial results in an *or*-wise manner (hence the name of the neuron). The concise notation underlines this flow of computing, $y=\mathrm{OR}(x;\ w)$ while the realization of the logic operations gives rise to the expression (commonly referred to as an s–t combination or s–t aggregation)

$$y_{\mathrm{OR}} = \mathop{\mathrm{S}}_{i=1}^{n}(x_i \mathrm{t} w_i) \tag{3}$$

Here "t" denotes a certain t-norm whereas "s" refers to some t-conorm (s-norm). Bearing in mind the interpretation of the logic connectives, the OR neuron realizes the following logic expression being viewed as an underlying logic description of the processing of the inputs

$$y_{\mathrm{OR}} = (x_1 \ and\ w_1)\ \ or\ \ (x_2\ and\ w_2)\ \ or \ldots or (x_n\ and\ w_n) \tag{4}$$

Apparently here the inputs are logically "weighted" by the values of the connections (w_i) before producing the final result. In other words we can treat "y" as a truth value of the above statement where the truth values of the inputs are affected by the corresponding weights. Noticeably, lower values of w_i discount the impact of the corresponding inputs; higher values of the connections (especially those being positioned close to 1) do not affect the original truth values of the inputs. In limit, if all connections w_i, $i=1, 2, \ldots, n$ are set to 1 then the neuron produces a plain *or*-combination of its inputs:

$$y_{\mathrm{OR}} = x_1\ or\ x_2\ or \ldots or\ x_n \tag{5}$$

The values of the connections set to zero eliminate the corresponding inputs. Computationally, the OR neuron exhibits nonlinear characteristics (that is inherently implied by the use of the t- and t-conorms).

AND neuron: the neurons in the category, denoted by $y=\mathrm{AND}(x;\ w)$ with x and w being defined as in case of the OR neuron, are governed by the expression

$$y_{\mathrm{AND}} = \mathop{\mathrm{T}}_{i=1}^{n}(x_i \mathrm{s} w_i) \tag{6}$$

In comparison with the previous category of the neurons, here the *or* and *and* connectives are used in a reversed order:

$$y_{\mathrm{AND}} = (x_1\ or\ w_1)\ and\ (x_2\ or\ w_2)\ and \ldots and\ (x_n\ or\ w_n) \tag{7}$$

first the inputs are combined with the use of the t-conorm and the partial results produced in this way are aggregated *and*-wise. Higher values of the connections reduce impact of the corresponding inputs. In limit $w_i=1$ eliminates the relevance of x_i. With all w_i set to 0, the output of the AND neuron is just an *and* aggregation of the inputs

$$y_{\mathrm{AND}} = x_1\ and\ x_2\ and \ldots and\ x_n \tag{8}$$

2.1. Additional functional features of logic neurons

While the above constructs could be viewed as generic logic neurons, there are two interesting and useful augmentations of the neurons which bring them even closer to the topologies of "standard" neurons existing in the realm of neurocomputing.

Incorporation of bias term: In analogy to the generic neuron as presented above, we could also consider a bias term, denoted by $w_0 \in [0, 1]$ which enters the processing formula of the fuzzy neuron in the following way

for the OR neuron

$$y_{\mathrm{OR}} = \mathop{\mathrm{S}}_{i=1}^{n}(x_i \mathrm{t} w_i) \mathrm{s} w_0 \tag{9}$$

for the AND neuron

$$y_{\mathrm{AND}} = \mathop{\mathrm{T}}_{i=1}^{n}(x_i \mathrm{s} w_i) \mathrm{t} w_0 \tag{10}$$

We can offer some useful interpretation of the bias by treating it as some nonzero initial truth value associated with the logic expression of the neuron. For the OR neuron it means that the output does not reach values lower than the assumed threshold. For the AND neuron equipped with some bias, we conclude that its output cannot exceed the value assumed by the bias. The question whether the bias is essential in the construct of the logic neurons cannot be fully answered in advance. Instead, we may include it into the structure of the neuron and carry out learning. Once its value has been obtained, its relevance could be established considering the specific value it has been produced during the learning. It may well be that the optimized value of the bias is close to zero for the OR neuron or close to one in the case of the AND neuron which indicates that it could be eliminated without exhibiting any substantial impact on the performance of the neuron.

Inhibitory functionality of logic neurons: Owing to the monotonicity of the t-norms and t-conorms, the computing realized by the neurons exhibits an excitatory character. This means that higher values of the inputs (x_i) contribute to the increase in the values of the output of the neuron. The inhibitory nature of computing realized by "standard" neurons by using negative values of the connections or the inputs is not available here as the truth values (membership grades) in fuzzy sets are confined to the unit interval. The inhibitory nature of processing can be accomplished by considering the complement of the original

input, $\overline{x}_i = 1 - x_i$. Hence when the values of x_i increase, the associated values of the complement decrease and subsequently in this configuration we could effectively treat such an input as having an inhibitory nature.

3. Logic processor in the processing of fuzzy logic functions: a canonical realization

The typical logic network that is at the center of logic processing originates from the two-valued logic and comes in the form of the well-known Shannon theorem of decomposition of Boolean functions. Let us recall that any Boolean function $\{0,1\}^n \rightarrow \{0,1\}$ can be represented as a logic sum of its corresponding minterms or a logic product of maxterms. By a minterm of "n" logic variables $x_1, x_2, \ldots, x_n$ we mean a logic product involving all these variables either in direct or complemented form. Having "n" variables we end up with 2^n minterms starting from the one involving all complemented variables and ending up at the logic product with all direct variables. Likewise by a maxterm we mean a logic sum of all variables or their complements. Now in virtue of the decomposition theorem, we note that the first representation scheme involves a two-layer network where the first layer consists of AND gates whose outputs are combined in a single OR gate. The converse topology occurs for the second decomposition mode: there is a single layer of OR gates followed by a single AND gate aggregating *or*-wise all partial results.

The proposed network (referred here as a logic processor–LP) generalizes this concept as shown in Fig. 1. The AND-OR mode of the logic processor comes with the two types of aggregative neurons being swapped between the layers. Here the first (hidden) layer is composed of the AND neuron and is followed by the output realized by means of the OR neuron.

The logic neurons generalize digital gates. The design of the network (viz. any fuzzy function) is realized through learning. If we confine ourselves to {0,1} values, the network's learning becomes an alternative to a standard digital design, especially a minimization of logic functions. The logic processor translates into a compound logic statement (we skip the connections of the neurons to emphasize the underlying logic content of the statement itself):

$$\text{if}(\text{input}_1 \; \textit{and} \ldots \textit{and} \; \text{input}_j) \; \textit{or} \; (\text{input}_d \; \textit{and} \ldots \textit{and} \; \text{input}_f) \; \text{then} \; \textit{output} \tag{11}$$

The LP's topology (and its underlying interpretation) is standard. Two LPs can vary in terms of the number of AND neurons, their connections but the format of the resulting logic expression is quite uniform (as a sum of generalized minterms).

It is worth noting that there is an infinite family of t- and s-norms which can be used in the realization of logic neurons. One has to be cognizant that some of these operators are more suitable for hardware realization. Let us consider some common cases in detail:

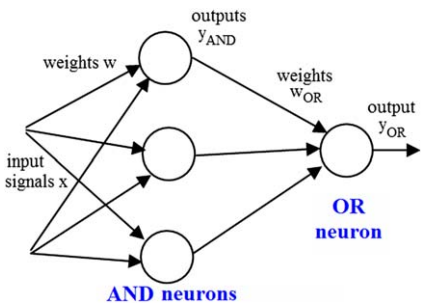

Fig. 1. A topology of the logic processor in its AND–OR mode.

(a) The minimum operator and maximum operator are easy to realize yet their input–output characteristics are limited. In particular, we observe a so-called lack of interactivity between the arguments. The operator exhibits an "extreme" behavior in the sense it picks up the most extreme argument (viz. the one with the minimal or maximal value). The result does not depend on the location (numeric values) of other arguments.

(b) the product operator is free from this shortcomings yet the multiplication operation is much more difficult to be realized in hardware. This problem is especially visible in case of many inputs like those shown in Eq. (7). In this case the multiplication would have to be repeated n times, while known electronic multipliers required to realize this task are either of low precision or become overly complex.

Given these computational and implementation requirements, we can arrive at conclusion that the Łukasiewicz connectives form an interesting alternative as being computationally flexible enough while still maintaining some features appealing from the implementation perspective. Łukasiewicz operators require only summation and subtraction operations, which makes them suitable for realization in analog current-mode technique. In this case even a large number of the input signals can be summed in a simple junction in a single processing step that simplifies the overall circuit.

Typical calculation scheme applied to Eqs. (4) and (7) is of iterative nature. This means that first we calculate either the *or* or the *and* operation using only two network inputs. Then the result of this operation is used as the input to the same operation at the second step together with the third network input. For n network inputs we need n *or* or *and* operations realized in a cascade manner. By using the Łukasiewicz operations, we reduce this calculation scheme to a single step only, since summation of all input signals represented by currents is realized in a junction in parallel. This significantly speeds up the calculations realized by such network and simplifies an overall structure of the circuit.

For illustrative purposes, Figs. 2 and 3 present input–output characteristics of the AND and OR neurons being implemented by means of the Łukasiewicz connectives. Note that the piecewise linear nature of the characteristics is highly visible while the detailed geometry of the neurons is implied by the values of the connections. Likewise we note a piecewise (and hence nonlinear) nature of the overall network.

4. Implementation of the OR and the AND neurons using classic CMOS fuzzy operators

Basic fuzzy logic operators together with their classic CMOS hardware implementation have been described in [15]. To realize these operators, the current-mode technique is the most suitable as it is suitable for an easy realization of summation and subtraction operations. The literature study shows that this technique is the most commonly used for this purpose [1,4,7,8]. The logic *or* and the *and* operators are also referred to as the bounded sum and the bounded product, respectively [15]. Using the approach presented in [15] we first propose the logic processor shown in Fig. 4. This figure presents a simplified diagram of the Łukasiewicz network that consists of two layers of neurons. In this case, the network comes in the AND-OR configuration, as shown in Fig. 1, although the dual configuration, that is OR–AND, arises as another alternative topology.

As mentioned above, the current-mode technique offers a very effective platform for the implementation of the logic operators, however it is a source of limited accuracy of these circuits. One of the main problems encountered in this realization is a mismatch

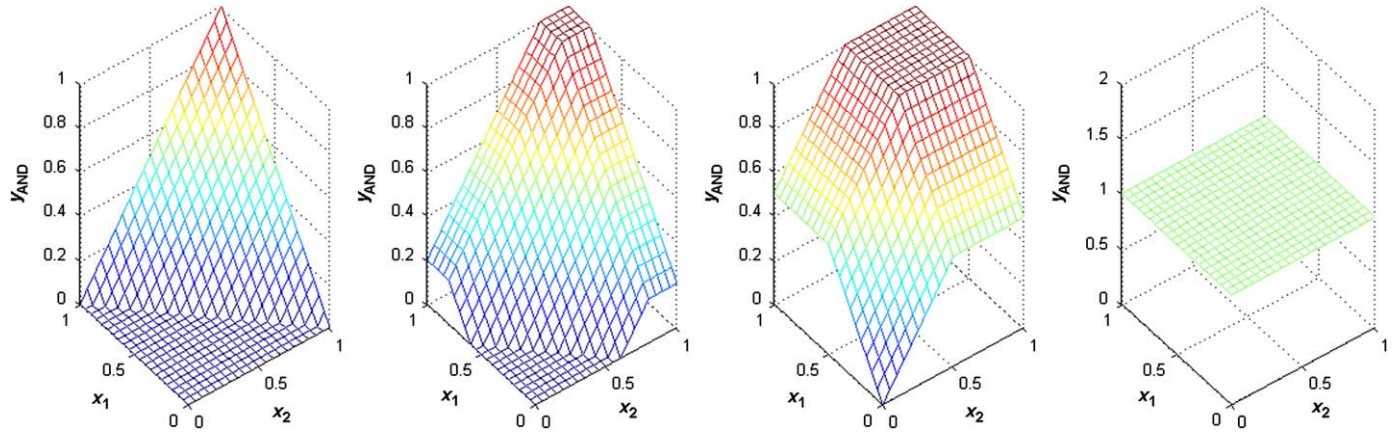

Fig. 2. Input–output characteristics of the AND neuron with 2 inputs (x_1 and x_2) and 2 corresponding weights; the cases from the left to the right are given for the following weights (w_1,w_2): [(0,0); (0.2,0.2); (0.5, 0.5); (1,1)].

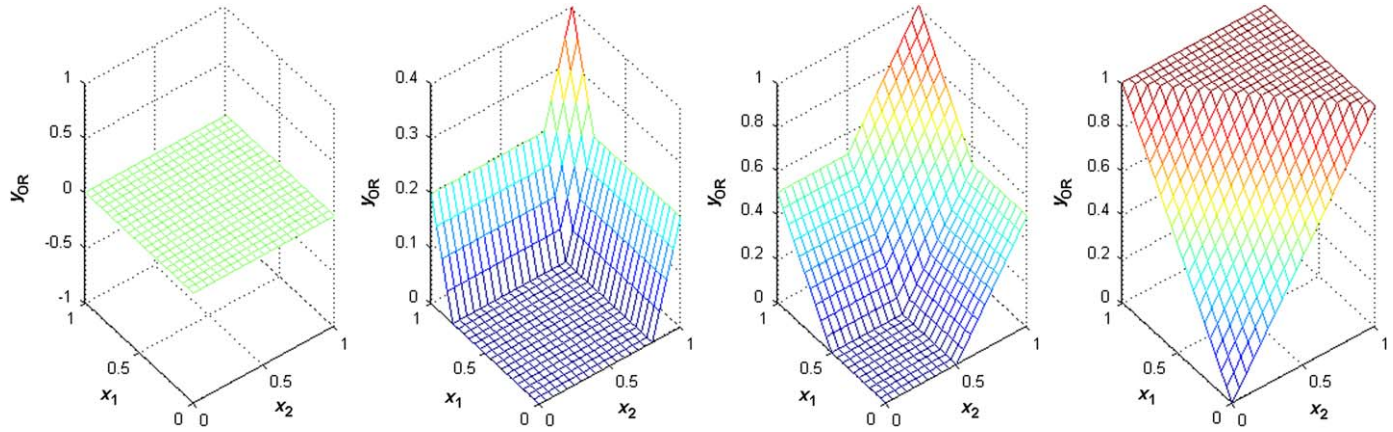

Fig. 3. Input–output characteristics of the OR neuron with 2 inputs (x_1 and x_2) and 2 corresponding weights; the cases from the left to the right are given for the following weights (w_1,w_2): [(0,0); (0.2,0.2); (0.5, 0.5); (1,1)].

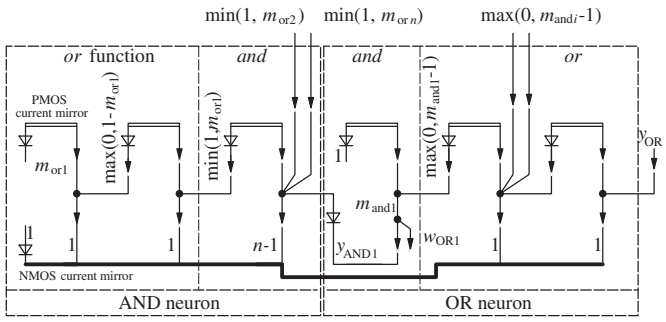

Fig. 4. Implementation of an example AND–OR Łukasiewicz network using Yamakawa's approach [14]. Terms $m_{or\ i}=x_i+w_i$ and $m_{and\ i}=y_{AND\ i}+w_{OR\ i}$ are used for simplifying the variables description.

between transistors present in the current mirrors [12]. Usually, the threshold voltage mismatch, ΔV_{th}, is assumed to be the main reason causing mismatch between the transistors working in a current mirror and therefore we limit our study to this parameter only. In practice several other mismatch components will further distort the signal. Assuming that both transistors have equal sizes then in the under threshold region the current mirror's gain that results from ΔV_{th} can be calculated as follows [2]:

$$\frac{I_2}{I_1} = e^{-\Delta V_{th}/nV_T} \tag{12}$$

while in the strong inversion region is expressed as

$$\frac{I_2}{I_1} = \frac{(V_{GS}-V_{TH}-\Delta V_{TH})^2}{(V_{GS}-V_{TH})^2} \tag{13}$$

where V_T is the thermal voltage (about 26 mV in the room temperature of 300 K). The influence of the transistor dimensions on the mismatch effect for different technologies has been reported and studied in many papers. The example diagram illustrating this phenomenon is shown in Fig. 5. For instance, in the CMOS 0.18 μm process, the standard deviation of the V_{th} mismatch varies in-between 0.7 and 2.9 [mV] given that the transistor gate's area ($W \cdot L$) varies in the range between 75 and 3 μm^2 [3].

In the under threshold region of operation, this effect introduces to the current mirror a gain error of 3% for large transistors; however this error could be as high as 11% for small transistors with an example gate area of 3 μm^2. This effect is illustrated in Fig. 6; refer to the upper curve. In the strong inversion region—the lower curve in Fig. 6—the influence of the mismatch on the gain is about 6–8 times smaller, but it is still important in case of circuits with large number of intermediate current mirrors like, for example, in the circuit shown in Fig. 4. In this case the number of current mirrors between the inputs and the output equals to 6. For large transistors with sizes of e.g. W/L=15/3 μm working in the strong inversion regime, the error of 10–15% is realistic, while in the under threshold region this error could be even as high as 40–60%. This makes the precision of the

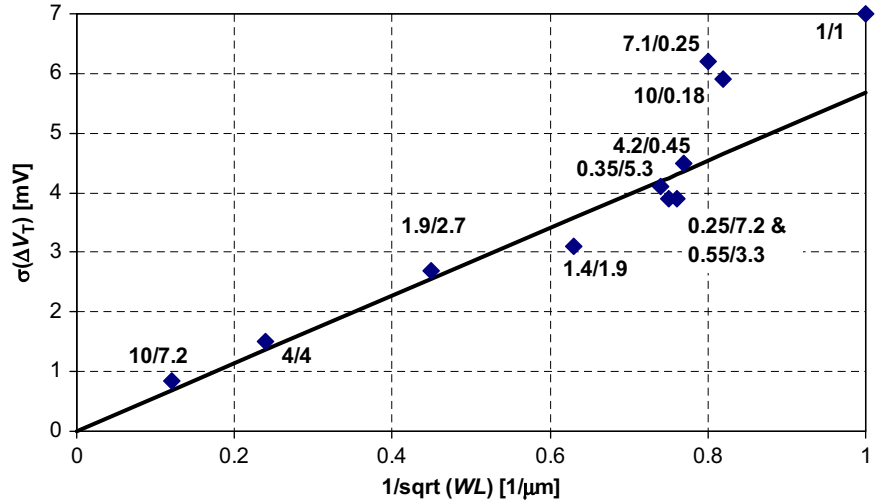

Fig. 5. Influence of transistor sizes on threshold voltage mismatch in CMOS 0.18 μm technology [2].

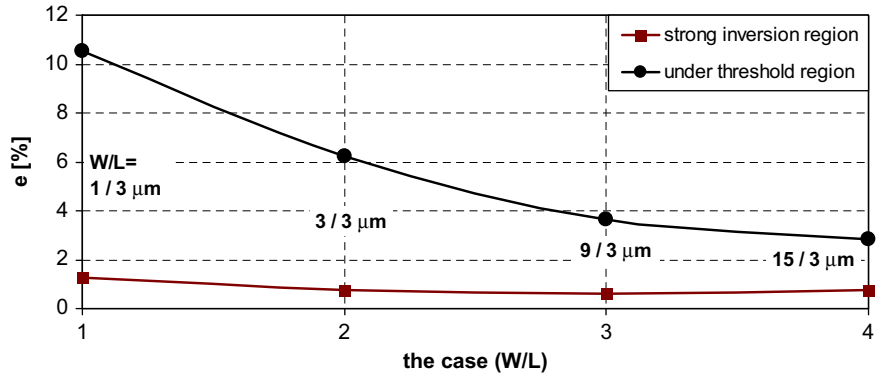

Fig. 6. Influence of transistor sizes on the gain error of a single current mirror in under threshold (upper curve) and strong inversion (bottom curve) regions, in CMOS 0.18 μm technology, for currents with an example constant value of 10 μA and the supply voltage V_{DD} of 1.8 V.

network based on the classic approach shown in Fig. 4 insufficient.

Fig. 6 illustrates another interesting phenomenon. The given values of the gain error are, for example, constant currents of 10 μA. In the strong inversion region this error depends not only on transistor sizes but also on the value of the gate to source voltage, V_{GS}, as shown in Eq. (13). Theoretically by increasing the transistor sizes we limit this error due to improved matching, but by keeping the currents at the same level we decrease the value of the V_{GS} voltage that increases the value of this error. As a result, both effects compensate one another thus resulting in almost constant value of this error seen in the bottom curve, with a minimum value for the aspect ratio (W/L) of 9/3 μm in this case. To make some further performance improvement in the strong inversion region possible it is necessary to increase the values of the currents, which will increase the value of the V_{GS} voltage. Unfortunately increasing the currents increases the power dissipation, so there is a trade-off between this parameter and the circuit precision.

To overcome the serious drawback described above, we propose the realization of new neurons described in the next section. The circuits built in this manner take advantage of the current mode, but the number of intermediate mirrors has been significantly reduced. In this case, signal paths between the inputs and the output of the circuit are controlled by simple current-mode comparators, realized using binary CMOS inverters and switches. This approach allows for significantly improved performance, since now the error of the overall network is limited to the error of a single current mirror only. Considering the results presented in Fig. 6, in the proposed network the aspect ratios (W/L) of 6/3 and 2/3 μm have been selected for the PMOS and the NMOS transistors respectively.

5. The proposed realization of the Łukasiewicz OR and AND logic neurons

5.1. OR neuron

The output of the multi-input OR neuron when implemented in software is commonly calculated in an iterative fashion by using the following expression:

$$y_{OR} = (((y_{and\ 1}\ or\ y_{and\ 2})\ or\ y_{and\ 3})\ or \ldots or\ y_{and\ n}) \quad (14)$$

where the outputs of the particular *and* functions are determined as follows:

$$y_{and\ i} = \max(0, x_i + w_i - 1) \quad (15)$$

Analyzing this expression from the left, the first *or* function in (14) returns either $y_{and\,1}+y_{and\,2}$ or 1, depending on which of these components assumes lower value. If $y_{and\,1}+y_{and\,2}$ is smaller then the next *or* function in the chain compares $y_{and\,1}+y_{and\,2}+y_{and\,3}$ with 1, returning the smaller of these terms. It can be shown that if at least one of these *or* functions returns 1 then all other *or*

functions also return 1. Since in hardware implementations many operations can be performed in parallel, therefore in this case it is much more convenient to use another scheme, which can be expressed as follows:

$$y_{OR} = \min(1, \sum_{i=1}^{n} y_{and\ i}) \tag{16}$$

(15) is very inconvenient to be implemented in the current-mode technique. The term after the coma can be either negative or positive when being compared with 0. Negative currents require much more complex circuits and therefore this expression must be transformed to avoid such a situation. The same result we obtain using the following formula:

$$y_{and\ i} = \max(1, x_i + w_i) - 1 \tag{17}$$

In this case we always compare only positive signals. Now the max function always returns the signal that is greater or equal to 1 and therefore subtraction of 1 in this equation never makes the output signal to be negative. Introducing new binary variables s_{ai} and s_{OR}, defined as

$$s_{ai} = \begin{cases} 1 & \text{for } x_i + w_i > 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad s_{OR} = \begin{cases} 1 & \text{for } \sum_{i=1}^{n} y_{and\ i} < 1 \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

(17) can be rewritten as

$$y_{and\ i} = s_{ai}(x_i + w_i - 1) \tag{19}$$

while (14) reads as follows:

$$y_{OR} = s_{OR} \sum_{i=1}^{n} s_{ai}(x_i + w_i - 1) + \overline{s_{OR}} \cdot 1 \tag{20}$$

(20) can be now much easier implemented using current-mode circuits. The variables s_{ai} and s_{OR} can be obtained from simple current-mode comparators realized as binary inverters. The resulting OR neuron is shown in Fig. 7. The ref signal visualized in this figure represents the value "1" in all formulas shown above. This current allows for scaling up and down all signals in this circuit, which is a very useful feature in case of systems working under different conditions.

The topology of this circuit minimizes the number of the intermediate current mirrors between the input and the output so that it limits an overall error. Note that the **and** blocks in Fig. 7 perform the operation described by (17). The comparators are used to compare the signals x_i+w_i signal with the reference current (ref), instead of x_i+w_i-1 with 0. Then reference current is then subtracted at the output of this block.

The important feature is that the input signals (x_i+w_i) are directly transferred to comparators as well as to the output branch, using the same multi-output current mirror. For this reason, the mismatch error in each branch is always related to the same input transistor and thus does not accumulate over successive processing steps. For sufficiently large transistors and large currents this error does not exceed 1–2%.

5.2. AND neuron

A very similar principle can be employed to construct the AND neuron. The calculation scheme realized in such neuron is similar to this of the OR neuron and is expressed as

$$y_{AND} = (((y_{or\ 1}\ and\ y_{or\ 2})\ and\ y_{or\ 3}) and\ \ldots\ and\ y_{or\ n}) \tag{21}$$

where

$$y_{or\ i} = \min(1, x_i + w_i) \tag{22}$$

If at least one of the *and* functions returns 0 then all other *and* functions also return 0, and therefore the relationship (21) can be rewritten as follows:

$$y_{AND} = \max\left(0, \sum_{i=1}^{n} y_{or\ i} - (n-1)\right) \tag{23}$$

Introducing the binary variables s_{oi} and s_{AND} defined as

$$s_{oi} = \begin{cases} 1 & \text{for } x_i + w_i < 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad s_{AND} = \begin{cases} 1 & \text{for } \sum_{i=1}^{n} y_{ori} + 1 > n \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

(22) can be rewritten in the following way:

$$y_{or\ i} = s_{oi}(x_i + w_i) + \overline{s_{oi}} \cdot 1 \tag{25}$$

while (23) reads as follows:

$$y_{AND} = s_{AND}[\sum_{i=1}^{n} s_{oi}(x_i + w_i) + \sum_{i=1}^{n} \overline{s_{oi}} \cdot 1 - n + 1] \tag{26}$$

Taking into account that

$$\overline{s_{oi}} \cdot 1 - 1 = -s_{oi} \cdot 1 \tag{27}$$

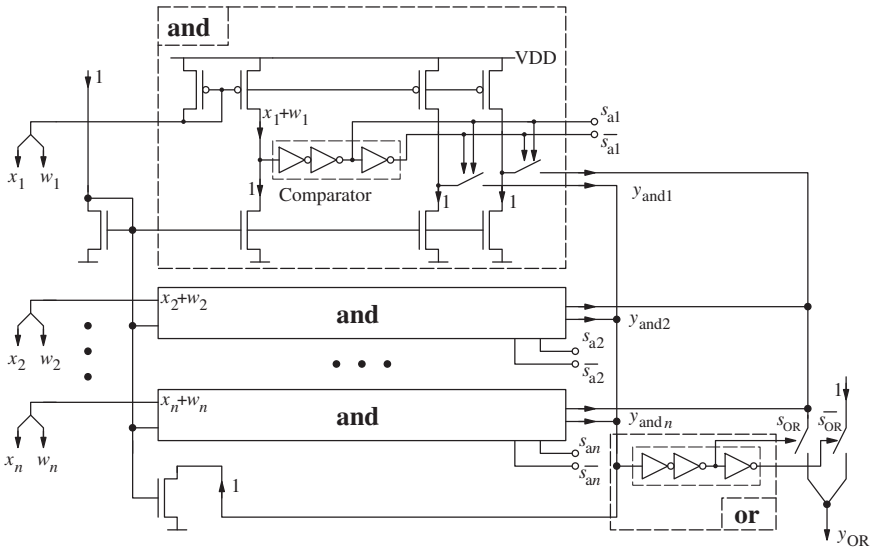

Fig. 7. The proposed Łukasiewicz OR neuron.

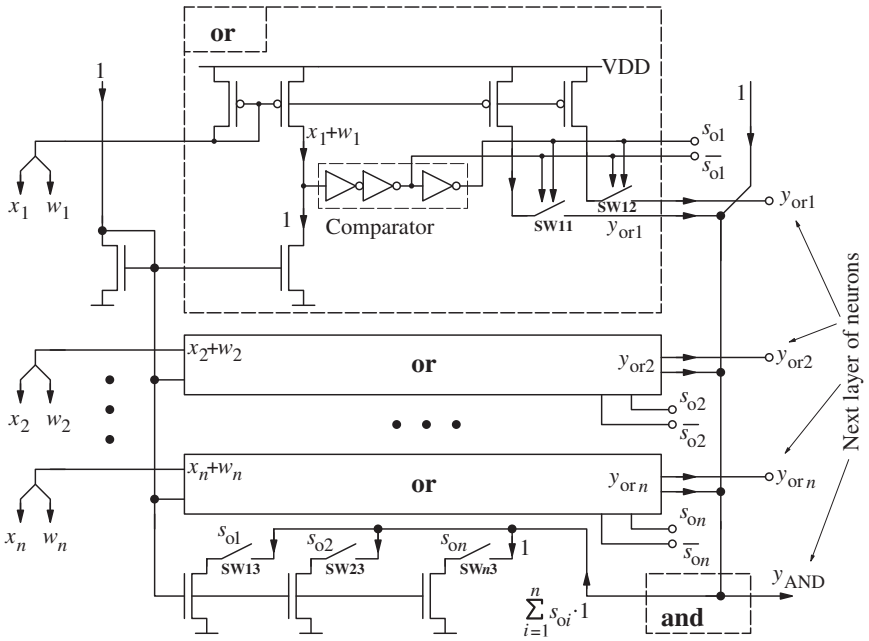

Fig. 8. The proposed Łukasiewicz AND neuron.

(26) can be simplified and rewritten as

$$y_{AND} = s_{AND}[\sum_{i=1}^{n} s_{oi}(x_i+w_i) - \sum_{i=1}^{n} s_{oi} \cdot 1 + 1] \tag{28}$$

The resulting structure of the AND neuron is illustrated in Fig. 8.

5.3. Implementation of two-layer Łukasiewicz logic network

The next step is to determine the hardware structure of entire two-layer network. Looking at expressions (14)–(16) and (21)–(23), one can note that a series of both the *or* and the *and* operations can be combined into a single function. Each layer in the proposed Łukasiewicz network contains different neurons, but at connections between both layers we have always the same operations i.e. *or–and–and–or* as well as *and–or–or–and* for the AND–OR and the OR–AND schemes respectively. This allows for combining operations of the same type, using the same principle like in Eqs. (16) and (23). As a result we obtain sequences: *or–and–or*; *and–or–and* for the AND–OR and OR–AND network schemes, respectively.

5.4. Realization of the AND–OR network

At the inputs of the second layers we have signals $y_{AND\,j}+w_{OR\,j}$, where $w_{OR\,j}$ are the weights of the output OR neuron. The output signal of the *j*th *and* function associated with the *j*th input in this neuron can be calculated as follows:

$$y_{AND\,j} = \max[0, \underbrace{\sum_{i=1}^{n} y_{or\,i,j} - n + 1}_{\text{output of the AND neuron}} + \underbrace{w_{OR\,j} - 1}_{\text{aggregation with the weight}}]$$
$$= \max(0, \sum_{i=1}^{n} y_{or\,i,j} - n + w_{OR\,j}) \tag{29}$$

or

$$y_{AND\,j} = s_{AND\,j}[\sum_{i=1}^{n} s_{oi,j}(x_i+w_{i,j}) - \sum_{i=1}^{n} s_{oi,j} \cdot 1 + w_{OR\,j}] \tag{30}$$

The output signal of the entire AND–OR network can be determined combining (16) and (30):

$$y_{OR} = \min(1, \sum_{j=1}^{m} y_{AND\,j}) = \min\Big(1, \sum_{j=1}^{m} s_{AND\,j}[\sum_{i=1}^{n} s_{oi,j}(x_i+w_{i,j}) - \sum_{i=1}^{n} s_{oi,j} \cdot 1 + w_{OR\,j}]\Big) \tag{31}$$

Using only logic variables, *s*, the output signal is expressed as follows:

$$y_{OR} = s_{OR} \sum_{j=1}^{m} s_{AND\,j}[\sum_{i=1}^{n} s_{oi,j}(x_i+w_{i,j}) - \sum_{i=1}^{n} s_{oi,j} \cdot 1 + w_{OR\,j}] + \overline{s_{OR}} \cdot 1 \tag{32}$$

The variables, *s*, are determined by comparators used on the *and* and the *or* layers in the network, the *m* parameter is the number of neurons in the network, while the *n* parameter is the number of the network inputs. To facilitate hardware implementation let us rewrite (32) in the following fashion:

$$y_{OR} = \sum_{j=1}^{m} \sum_{i=1}^{n} s_{OR} s_{AND\,j} s_{oi,j}(x_i+w_{i,j}) - \sum_{j=1}^{m} \sum_{i=1}^{n} s_{OR} s_{AND\,j} s_{oi,j} 1 + \sum_{j=1}^{m} s_{OR} s_{AND\,j} w_{OR\,j} + \overline{s_{OR}} \cdot 1 \tag{33}$$

where

$$s_{AND\,j} = \begin{cases} 1 & \text{for } \sum_{i=1}^{n} s_{oi,j}(x_i+w_{i,j}) + w_{OR\,j} > \sum_{i=1}^{n} s_{oi,j} \cdot 1 \\ 0 & \text{otherwise} \end{cases}$$
$$s_{OR} = \begin{cases} 1 & \text{for } \sum_{j=1}^{m} y_{AND\,j} < \sum_{i=1}^{n} \sum_{j=1}^{m} s_{oi,j} s_{AND\,j} \cdot 1 \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

The structure of the resultant AND–OR network is shown in Fig. 9. Products of particular logic variables, $s_{OR} \cdot s_{AND} \cdot s_{or}$ and $s_{OR} \cdot s_{AND}$ are calculated using standard digital 2-input AND gates. The important advantage of this realization is that all analog signals, i.e. $x_i+w_{i,j}$, w_j

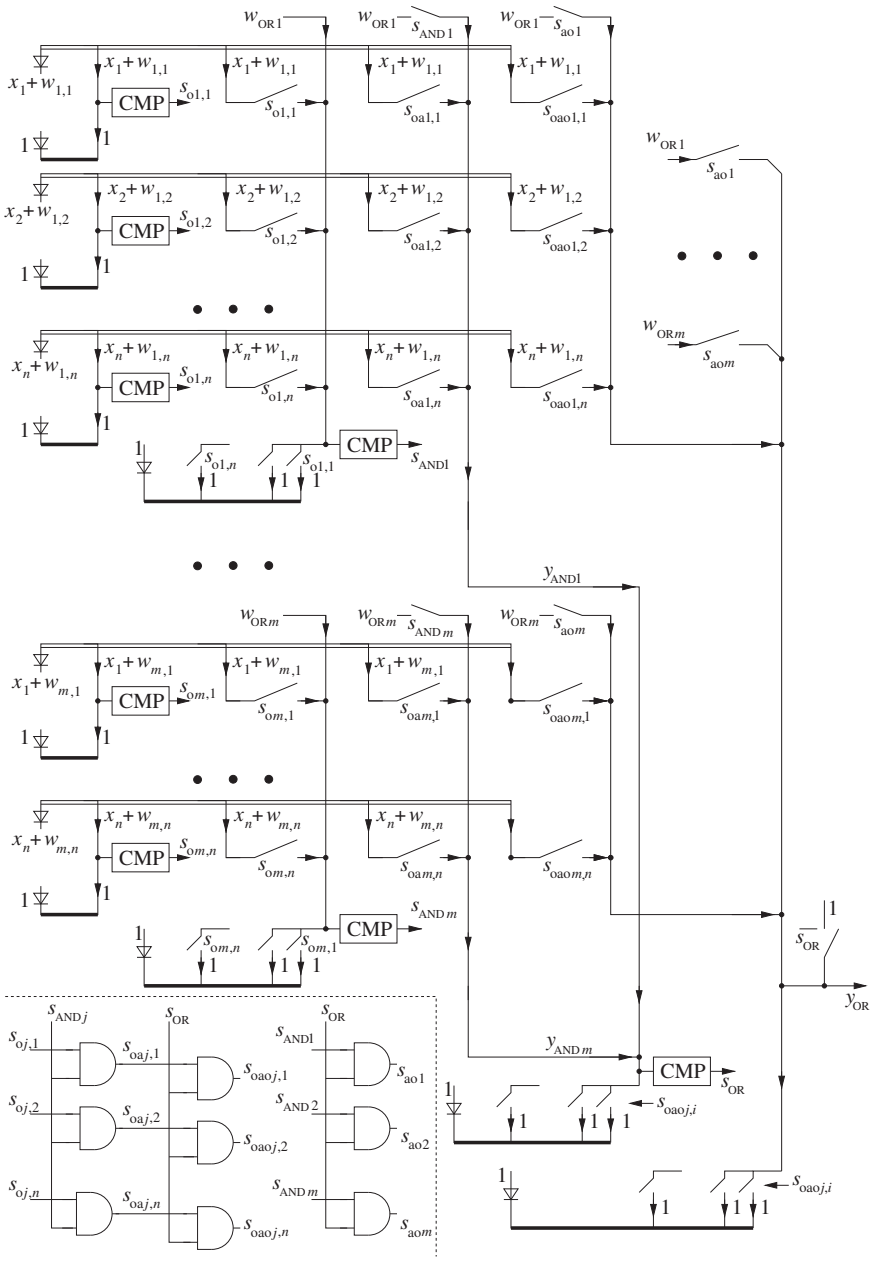

Fig. 9. The proposed analog current-mode Łukasiewicz AND–OR network.

and the reference currents represented by 1, are copied at the network output using only single current mirrors and single switches, which are controlled by the digital AND gates. As a result, the precision of this circuit is approximately equal to the precision of a single current mirror, which is one of the main advantages of the proposed solution.

The input signals x as well as the weights w are always copied using only PMOS-type current mirrors, while the reference currents ref always using only the NMOS-type mirrors i.e. always under the same conditions.

Some disadvantage of this implementation, in comparison with the circuit shown in Fig. 4, is the necessity of using digital elements, which increase complexity of the circuit. This problem is not very essential though, since the number of these gates, $n\cdot(m+1)$, is not very large and typically does not exceed several dozen to several hundreds. As a result, the chip area does not increase significantly, since transistors in these gates are designed in such a way so that they exhibit minimal sizes, in contrary to transistors used in analog signal paths. Each of the input pairs $x_i+w_{i,j}$ is copied 4 times (5 transistors), while each of the $w_{\mathrm{OR}j}$ weights is copied 3 times (4 transistors). Three (two in case of w_{OR} signals) of these copies are provided to particular layers in the network, while the 4th (3rd) copy at the network output. The number of transistors in analog signal paths, excluding switches realized as transmission gates, approximately equals $3\cdot m\cdot(3n+1)$. In comparison with transistors used in digital elements, these transistors have much larger sizes. As a result, the digital part of the chip occupies only some fraction of the total chip area.

5.5. OR–AND network

At the inputs of the output AND neuron we have the signals $y_{\mathrm{OR}j}$ coming from the first layer. Taking into account the principle that several *or* operations can be combined into one, the $y_{\mathrm{OR}j}$ signals already contain weights of the AND neuron, $w_{\mathrm{AND}j}$. These

signals are described as follows:

$$y_{OR\,j} = s_{OR\,j}\Big(\sum_{i=1}^{n} s_{ai,j}(x_i + w_{i,j} - 1) + w_{AND\,j}\Big) + \overline{s_{OR\,j}} \cdot 1 \tag{35}$$

Using (23) and (35), the output of the entire OR–AND network can be described as follows:

$$y_{AND} = \max\Big(0, \sum_{j=1}^{m}\Big(s_{OR\,j}\Big(\sum_{i=1}^{n} s_{ai,j}(x_i + w_{i,j} - 1) + w_{AND\,j}\Big) + \overline{s_{OR\,j}} \cdot 1\Big) - (m-1)\Big) \tag{36}$$

Using (26) we can rewrite the above expression as follows:

$$y_{AND} = s_{AND}[\sum_{j=1}^{m}\Big(s_{OR\,j}\Big(\sum_{i=1}^{n} s_{ai,j}(x_i + w_{i,j} - 1) + w_{AND\,j}\Big) + \overline{s_{OR\,j}} \cdot 1\Big) - (m-1)] \tag{37}$$

Finally, we obtain:

$$y_{AND} = \sum_{j=1}^{m}\sum_{i=1}^{n} s_{AND} s_{OR\,j} s_{ai,j}(x_i + w_{i,j} - 1) + \sum_{j=1}^{m} s_{AND} s_{OR\,j} w_{AND\,j} + \sum_{j=1}^{m} s_{AND}\overline{s_{OR\,j}} \cdot 1 - s_{AND}(m-1) \tag{38}$$

which using (27) can be rewritten as follows:

$$y_{AND} = \sum_{j=1}^{m}\sum_{i=1}^{n} s_{AND} s_{OR\,j} s_{ai,j}(x_i + w_{i,j} - 1) + \sum_{j=1}^{m} s_{AND} s_{OR\,j} w_{AND\,j} - \sum_{j=1}^{m} s_{AND} s_{OR\,j} \cdot 1 + s_{AND} 1 \tag{39}$$

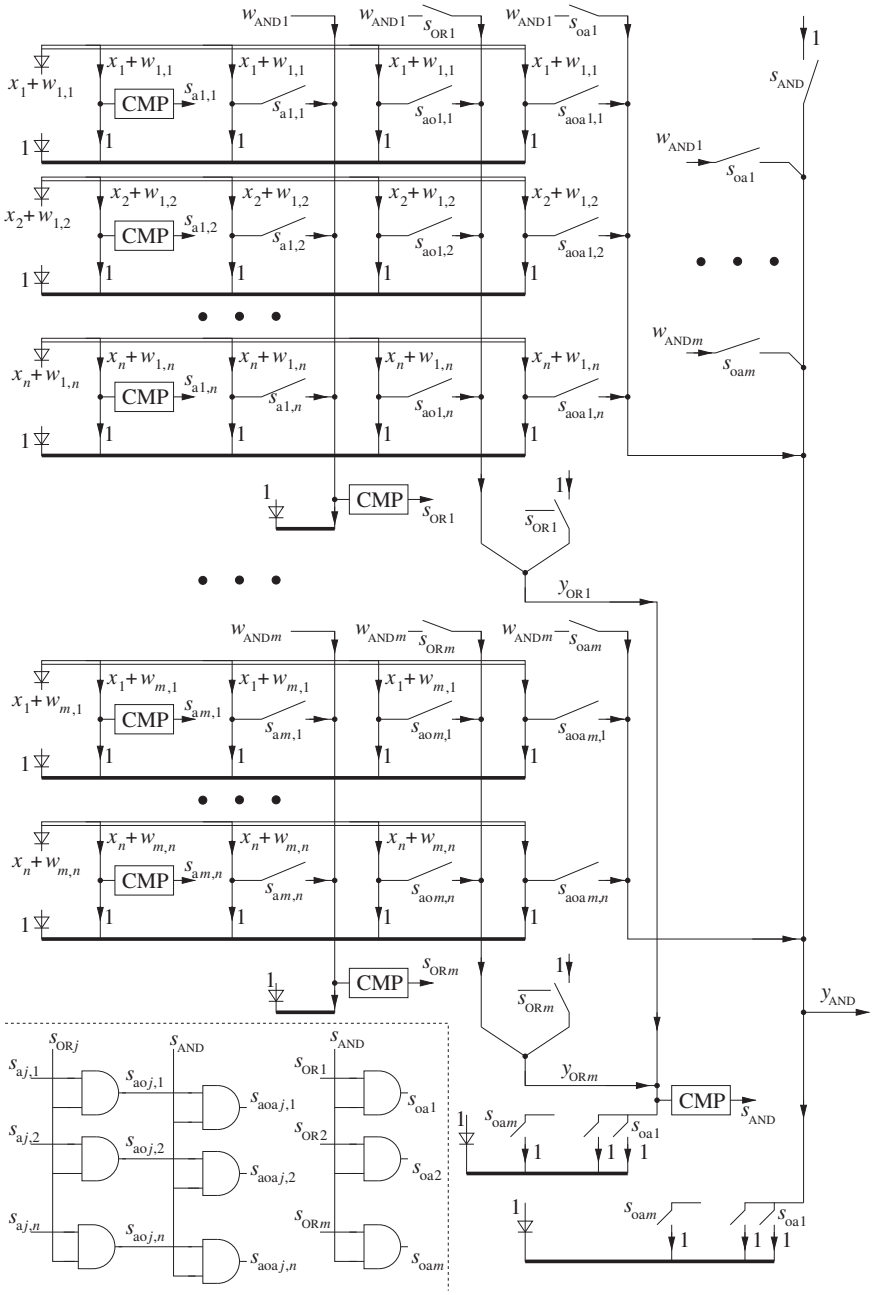

Fig. 10. The proposed analog Łukasiewicz OR–AND network.

In this case, the logic variables, s_{ORj} and s_{AND} shown in (39) are computed as follows:

$$s_{ORj} = \begin{cases} 1 & \text{for } \sum_{i=1}^{n} s_{ai,j}(x_i + w_{i,j} - 1) + w_{ANDj} < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$s_{AND} = \begin{cases} 1 & \text{for } \sum_{j=1}^{m} y_{ORj} + 1 > m \\ 0 & \text{otherwise} \end{cases} \quad (40)$$

The resulting structure of the OR–AND network is given in Fig. 10.

5.6. Verification of the proposed circuits

Illustrative simulation results obtained for the logic OR and AND neurons implemented using the proposed approach are shown in Figs. 11 and 12, respectively. In the completed simulations, both neurons have 4 inputs (x) and 4 weights (w). For the ease of visualization, the values of x signals are kept the same. The same holds for the input weight signals w. The upper plot in both figures presents the input and the output analog currents. Two example cycles are shown for better illustration. The first cycle starts at 200 ns and ends at 1900 ns, while the second one starts at 2100 ns and ends at 3100 ns.

In Fig. 11, in the first cycle, all the x and the w input signals have equal values. These signals start rising at 1200 ns. At this time the outputs of the comparators, s_{ai}, in the first layer i.e. in the *and* operations in Fig. 7 assume logical "0", since the particular sums x_i+w_i are smaller than the reference current I_{ref}, which plays a role of the "1" signal in the equations shown above. As a result, the switches SW1 and SW2 are opened and the corresponding $y_{and\,i}$ signals are zero, which is in the agreement with (15). When the x and the w signals become equal to $0.5 \cdot I_{ref}$, the s_{ai} signals become logical "1" and the $y_{and\,i}$ signals follow the values $x_i+w_i-I_{ref}$, as given by expression (19). In the period between 1300 ns and c.1330 ns the sum of all the $y_{and\,i}$ signals is less than the I_{ref} current, resulting in the s_{OR} signal to be equal to logical "1", as given by (18), and the neuron output signal, y_{OR}, follows the sum of the four $y_{and\,i}$ signals, as given by (20). At the point A, all $y_{and\,i}$ signals become equal to $0.25 \cdot I_{ref}$, the y_{OR} signal which is sum of the $y_{and\,i}$ signals becomes equal to I_{ref}, and the comparator output at the second layer s_{OR} becomes logical "0". As a result, the neuron output y_{OR} is switched directly to the reference signal, while the switch controlled by the s_{OR} signal is in this period opened. This is the reason why the $y_{and\,i}$ signals become zero. This is worth noticing that each *and* operation provides two copies of the $y_{and\,i}$ signal and only the copy that flows to the neuron output is zeroed. The copy that is provided to the comparator input in the *or* operation is still nonzero, as expected.

In the second cycle the x signals differs from w signals. In the time moment 2300 ns the x signal reaches the maximum value and equals I_{ref}. Since the neuron weights equal zero in this time therefore both the $y_{and\,i}$ signals and the neuron output are zero. This situation corresponds with the situation shown in Fig. 3 (the left diagram). In the time moment 2400 ns the weights start rising and when their values become 200 nA and the s_{ai} signals become logical "1", the $y_{and\,i}$ signals start following the values $x_i+w_i-I_{ref}$.

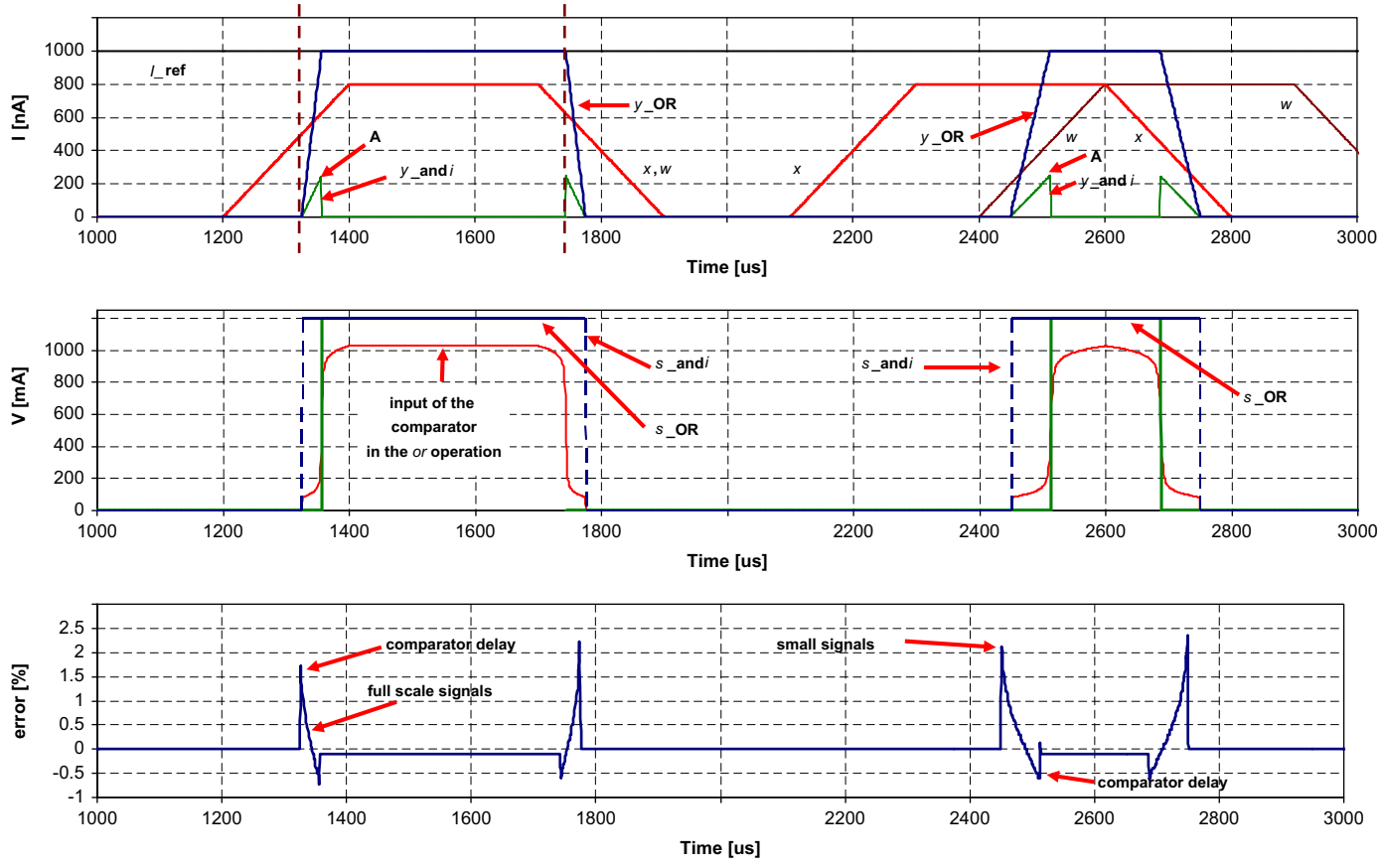

Fig. 11. Simulation results for OR neuron: (top) input x, weight w and the output signals, (middle) operation of the comparator (bottom) error between theoretical and the simulated cases.

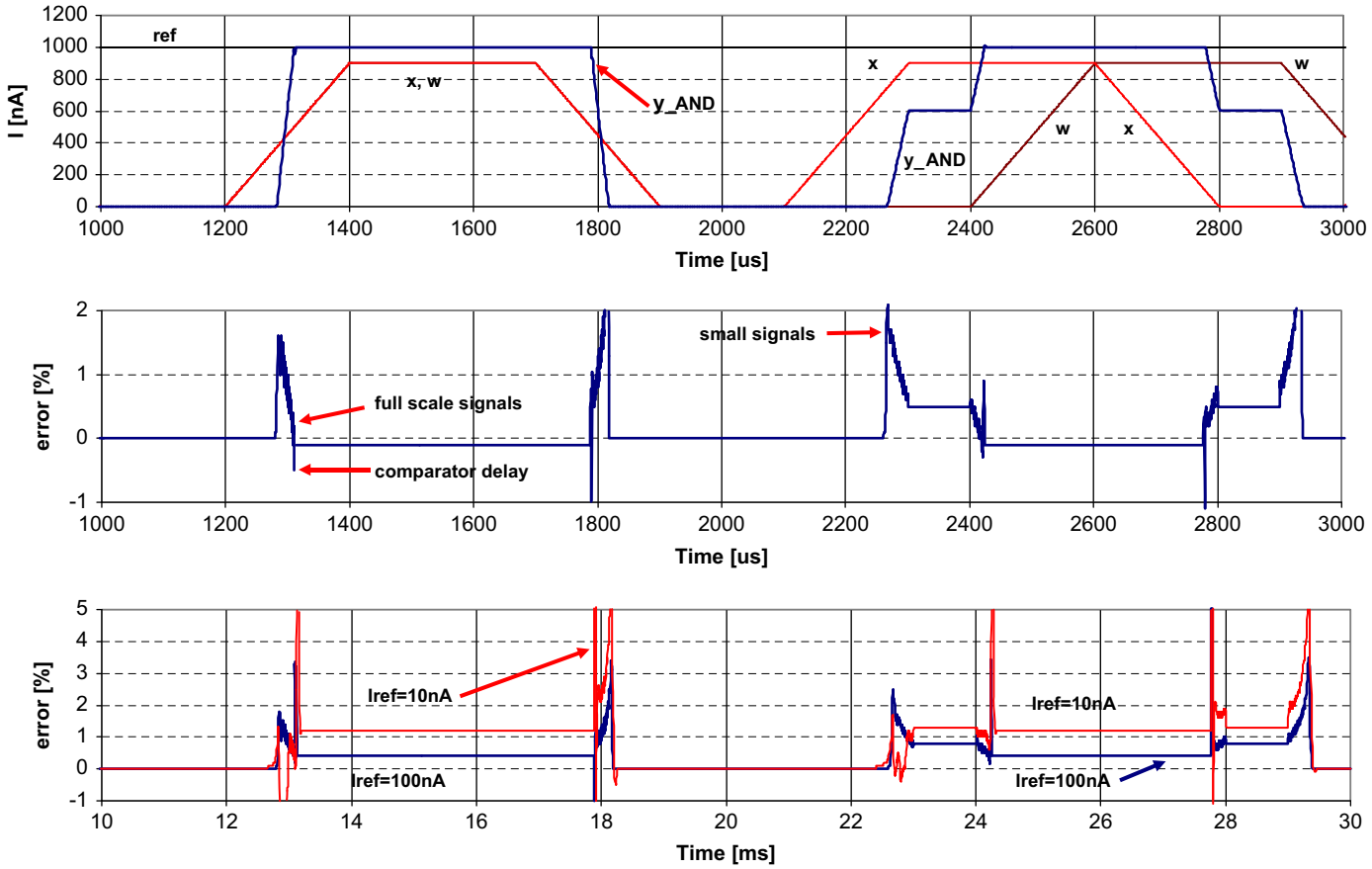

Fig. 12. Simulation results for the AND neuron: (top) input and output signals for the I_{ref} current (the "1" signal) of 1 μA, (middle) resultant error for this current, (bottom) errors for I_{ref} of 100 and 10 nA.

The second plot in Fig. 11 illustrates an operation of the comparators both in the first and in the second layer of the OR neuron.

A very important aspect here concerns a precision of the designed circuits, which has been evaluated by comparing the signals obtained in transistor level simulations with theoretical values obtained from (4) and (6) for the same inputs. The third plot in Fig. 11 presents the percentage error for the OR neuron. The error is the highest for small signals. It is also quite high in periods, in which comparators have to change the state which introduces some delay. For the full scale input signals the error is well below 0.2%, although this error can be potentially increased by 1–2% owing to the mismatch effect described above.

The similar results, Fig. 12, have been obtained for the AND neuron. For small values of the x_i+w_i signals being lower than the I_{ref} current the comparators outputs, s_{oi}, at the first layer i.e. in the or operations are logical "1" (Eq. (24)). The switches SWi1, SWi2, SWi3 are closed and the y_{ori} signals follow the signals x_i+w_i. In the first cycle that starts at 1200 ns the neuron output signal, y_{AND}, starts rising when both the x and the w signals, which are equal in this period, equal $0.375 \cdot I_{ref}$. For this given value the first term in (28) i.e., the sum of the $s_{oi} \cdots (x_i+w_i)$ signals equals 3000 nA, the second term equals 4000 nA and the total signal in brackets equals 0 ns. In the second cycle the x and the w signals are different. The x signal start rising as first and at 2300 ns reaches the value of 900 nA. As a result the y_{AND} signal becomes equal to $4 \cdot 900 - 4000 + 1000\,[\text{nA}] = 600\,\text{nA}$. Further increase of the y_{AND} signal starts when the w_i signals start rising at 2400 ns. When the x_i+w_i signals become larger than I_{ref} current, the variables s_{oi} become logical "0" and the switches SWi1–SWi3 in particular *or* operations are opened. For all switches being opened only the reference current flows to the output of the neuron.

In the AND neuron, the error calculated in relation to the full scale signal is at the level of 0.25%—the middle plot. The third plot presents the error in case of reduced values of the input signals, namely for I_{ref}=100 and 10 nA. We note that although the error increases for smaller signals, this increase is relatively limited.

One of the key objectives is to reduce power dissipation of electronic circuits to enable their effective usage in ultra low power portable devices [14]. The ability of the circuits to work with signals which vary in a large range of values is one of the paramount features of our design. In this type of circuits the power dissipation almost linearly depends on the values of the input signals. For the input signals varying in the range up to 1 μA the average power dissipation equals to c.a. 10 μW, while for the range of 10 nA the power dissipation becomes reduced to 95 nW only.

The results presented in this study are reported for the CMOS 0.18 μm process and the supply voltage of 1.2 V. One could note that these circuits operate properly between 1.8 and 0.9 V of the supply. In newer CMOS technologies, the supply voltage could be further reduced further facilitating the reduction of power dissipation.

The networks shown in Figs. 9 and 10 composed of 4–6 neurons and with 4 inputs have been verified in Hspice simulations in the similar way like single AND and OR neurons. The results concerning the circuit precision are comparable.

One of the important questions being formed in case of the realization of integrated systems is the chip area, which has a significant influence on the cost of the device. Taking into account other chips previously realized be the authors in the CMOS 0.18 μm technology, including current-mode circuits, e.g. [5,13],

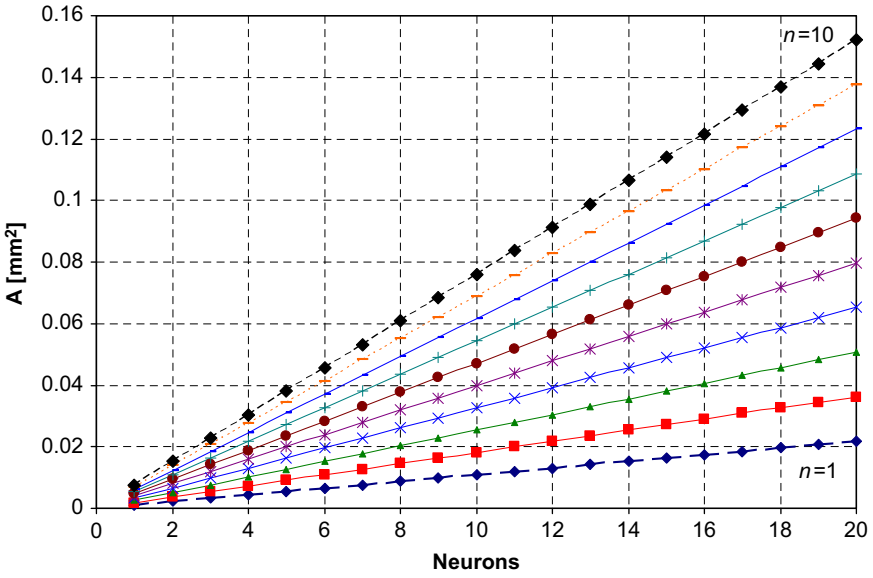

Fig. 13. Evaluation of the chip area as a function of the number of neurons (m) and the number of the network inputs (n) in case of CMOS 0.18 μm technology for the OR–AND network. The results for the AND–OR network are very similar.

the chip area of the proposed circuit has been evaluated as a function of the number of neurons (m) and the number of inputs (n) in the network. The results are shown in Fig. 13.

The presented results have been obtained on the basis of the number of transistors in the network, as a function of the n and the m variables. Since the digital blocks require only small transistors, while the analog blocks use larger transistor to minimize the mismatch problem described above, both types of transistors have been counted separately and multiplied by areas for particular types of transistors. Some area has been assumed for the connecting paths between particular blocks in the chip. Finally, the obtained areas have been multiplied by a correction factor, which has been calculated on the basis of the projects realized earlier by the authors. The presented areas do not include the adaptation mechanism, which is not within the scope of this paper. Such mechanism has been designed by the authors in another project i.e. the analog Kohonen neural network [5,13] and its area in CMOS 0.18 μm technology equals 1000 μm^2. The number of such blocks in the network proposed in this paper is equal to $n \cdot m$. Fig. 13 shows that even in case of very advanced system with 20 neurons and 10 inputs the chip area will be relatively small and equal to c. 0.15 mm^2 without the adaptation mechanism or c. 0.35 mm^2 with the adaptation mechanism.

This is also worth noting that all the x_i+w_i signals are always copied to comparators and to the network output using the PMOS current mirrors, while the reference current I_{ref} always using the NMOS mirrors. If an offset in comparators occurs and this offset has a similar value for all comparators, which is possible in a careful layout design, it can be compensated by adjustment of the value of reference current, which is an additional advantage here.

6. Conclusions

In this study, we have proposed the new ultra low power circuit realization of *or* and the *and* Łukasiewicz operators used as logic connectives in OR and AND neurons and neural networks. In the proposed circuits the error reduction for small signals are achieved by eliminating a chain of current mirrors between the input and the output of the neurons. The simulation results show that the circuits can operate with currents that are at the level of single nAmps, which is an attractive feature especially for very low power portable devices.

In the present work, we have not described the adaptation mechanism realized at the transistor level that would be suitable for the proposed network. The implementation of such mechanism becomes the problem by itself. We have proposed the mechanism of this nature earlier when dealing with analog Kohonen self organized feature maps [5,13]. After some modifications, this mechanism can be adopted for the proposed network.

The implementation of the bias term, as described in Section 2, has not been discussed here but it can be easily realized in hardware, by simple addition of a single branch at the output of each neuron in the network.

References

[1] C.-Y. Chen, Y.-T. Hsieh, B.-D. Liu, Circuit implementation of linguistic-hedge fuzzy logic controller in current-mode approach, IEEE Transactions on Fuzzy Systems 11 (5) (2003) 624–646.
[2] M. Conti, G.D. Betta, S. Orcioni, G. Soncini, C. Turchetti, N. Zorzi, Test structure for mismatch characterization of MOS transistors in subthreshold regime, IEEE International Conference on Microelectronic Test Structures 10 (1997) 173–178.
[3] J.A. Croon, M. Rosmeulen, S. Decoutere, W. Sansen, H.E. Maes, An Easy-to-Use mismatch model for the MOS transistor, IEEE Journal of Solid-State Circuits 37 (8) (2002) 1056–1064.
[4] E. Farshidi, A low-power current-mode defuzzifier for fuzzy logic controllers, International Conference on Signals, Circuits and Systems (SCS) (2008) 1–4.
[5] R. Długosz, T. Talaska, J. Dalecki, R. Wojtyna, Experimental Kohonen neural network implemented in CMOS 0.18μm technology, in: 15th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES), Poznan, Poland, 2008, pp. 243–248.
[6] J. Jang, C. Sun, E. Mizutani, Neuro-Fuzzy and Soft Computing, Prentice Hall, Upper Saddle River, NJ, 1997.
[7] J.W. Mills, Area-efficient implication circuits for very dense Lukasiewicz logic arrays, International Symposium on Multiple-Valued Logic, 1992, pp. 291–298.
[8] A. Morgul, T. Temel, Current-mode level restoration: circuit for multi-valued logic, Electronics Letters 41 (5) (2005) 230–231.
[9] W. Pedrycz, Fuzzy neural networks and neurocomputations, Fuzzy Sets and Systems 56 (1993) 1–28.
[10] W. Pedrycz, Heterogeneous fuzzy logic networks: fundamentals and development studies, IEEE Transactions on Neural Networks 15 (2004) 1466–1481.
[11] W. Pedrycz, A. Rocha, Knowledge-based neural networks, IEEE Transactions on Fuzzy Systems 1 (1993) 254–266.
[12] A. Rodriguez-Vazquez, R. Navas, M. Delgado-Restituto, F. Vidal-Verdu, A modular programmable CMOS analog fuzzy controller chip, IEEE Transactions

on Circuits and Systems II: Analog and Digital Signal Processing 46 (3) (1999) 251–265.
[13] T. Talaśka, R. Długosz, W. Pedrycz, Adaptive weight change mechanism for Kohonens's neural network implemented in CMOS 0.18 μm technology, in: 11th European Symposium on Artificial Neural Networks, Bruges, Belgium, 2007, pp. 151–156.
[14] J.X. Xu, C. Xue, C.C. Hang, K.V. Palem, A fuzzy control chip based on probabilistic CMOS technology, IEEE International Conference on Fuzzy Systems FUZZ-IEEE (2008) 174–179.
[15] T. Yamakawa, T. Miki, The current mode fuzzy logic integrated circuits fabricated by standard CMOS process, IEEE Transactions on Computers C 35 (2) (1986) 161–167.

Rafał Długosz is currently with the Swiss Federal Institute of Technology in Lausanne (EPFL) in Switzerland, Institute of Microtechnology (IMT). He received M.Sc. in automatic and robotic in 1996 and Ph.D. in telecommunication in 2004 (with honours), both from Poznań University of Technology (PUT) in Poland. From 1996 to 2001 he was with the Institute of Electronics and Telecommunication at PUT. From 2001 he is with Department of Computer Science and Management at PUT. He is the fellow of several scientific fellowships from Foundation for Polish Science in Poland (for young scientists and for young doctors) and from European funds (Marie Curie Outgoing International Fellowship). Within these fellowships between 2005 and 2008 he was with Department of Electrical and Computer Engineering at the University of Alberta in Canada and since 2006 he holds the scientist position in EPFL. His main research areas are ultra low power reconfigurable analog and mixed analog-digital integrated circuits such as analog finite impulse response filters, analog-to-digital converters and neural networks. Dr. Długosz has been involved in numerous projects granted by EU and Polish Government. He has published over 80 research papers, including several book chapters.

Witold Pedrycz (M'88-SM'90-F'99) received the M.Sc., and Ph.D., D.Sci. all from the Silesian University of Technology, Gliwice, Poland. He is a Professor and Canada Research Chair (CRC) in Computational Intelligence in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada. He is also with the Polish Academy of Sciences, Systems Research Institute, Warsaw, Poland.

His research interests encompass computational intelligence, fuzzy modeling, knowledge discovery and data mining, fuzzy control including fuzzy controllers, pattern recognition, knowledge-based neural networks, granular and relational computing, and Software Engineering. He has published numerous papers in these areas. He is also an author of 12 research monographs. Witold Pedrycz has been a member of numerous program committees of IEEE conferences in the area of fuzzy sets and neurocomputing. He serves as Editor-in-Chief of IEEE Transactions on Systems Man and Cybernetics-part A and Associate Editor of IEEE Transactions on Fuzzy Systems. He is also an Editor-in-Chief of Information Sciences. Dr. Pedrycz is a recipient of the prestigious Norbert Wiener award from the IEEE Society of Systems, Man, and Cybernetics and an IEEE Canada Silver Medal in Computer Engineering.