Threshold Logic Circuits Implementation Using FPAA

Boyan P. Lyubenov[1], Emil D. Manolov[2]

Abstract – **The paper presents the results from the design and investigation of basic threshold logic circuits using Field Programmable Analog Array (FPAA) of Anadigm Inc. To this aim, the functional model of the threshold logic gate is discussed and some approaches in building logic functions are presented. The results from implementation of different FPAA variants of threshold logic structures are described. The use of the FPAA ensures possibilities for simple programming and dynamic reconfiguration of different values of the weights on the inputs as well as a flexible realization of different logic functions. The results clearly present the practical use of the discussed approach and could find application for fast prototyping of threshold logic based systems with possibilities for flexible real-time programming and reconfiguration of the parameters and functions.**

Keywords – **Threshold logic, Neural networks, Programmable analog circuits, Field Programmable Analog Array, FPAA**

I. Introduction

The threshold logic (TL) concept was introduced as theory of logic gates [1], [2], and over the years has promised much in terms of reduced logic depth and gate count, compared to conventional Boolean logic based design. Since the basic TL-gate is functionally more powerful than those of the conventional logic, many complex functions can be synthesized in TL with lesser number of gates in a shorter logic depth. Despite the theoretically obvious merits, TL has never had a significant impact in practice, most probably due to the lack of efficient physical realization.

In the last decade the fast development of VLSI technology has made neurocomputer design not only a research topic but several chips have been developed [3], [4], [5]. Research on hardware implementations of neurons has recently been very active. In [6] and [7], for fast prototyping of neural networks, is used Field Programmable Analog Array (FPAA).

The FPAA circuits have been introduced in 1990's by some of the biggest chips' suppliers. They are analog equivalents of the Field Programmable Gate Array (FPGA). The main reason behind FPAA was to help analogue designer to debug their systems long before real silicon comes out the fabs so that significant time-to-market reduction can be achieved. Nowadays, the FPAA technology is very flexible and powerful technology for fast prototyping of different electronic systems [8].

In the presented research is used FPAA chip introduced by Anadigm® [9]. The Anadigm® AN220E04 is a reconfigurable analog device based on switched capacitor technology.

The paper demonstrates a practical approach for building, simulation, implementation and verification of different threshold logic circuits using FPAA.

II. Basic Threshold Logic Theory

A threshold gate is defined as an n-input logic gate, functionally similar to a hard-limiting neuron without learning capability [1]. The gate takes n binary inputs x_1, x_2 … x_n and provides a single binary output y as it is shown in Fig. 1.

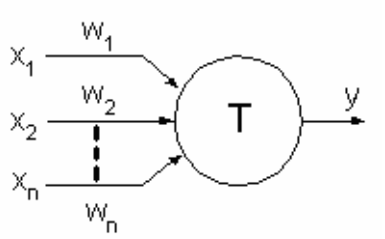

Fig. 1. Threshold logic gate

The output of the gate is determined by the following set of relations:

$$y = \begin{cases} 1, & if \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & otherwise \end{cases}, \qquad (1)$$

where T is the threshold and the w_i is the weigh associated with the i-th input variable x_i. The function can be written in more compact form:

$$y = \mathrm{sgn}\left(\sum_{i=1}^{n} w_i x_i - T \right), \qquad (2)$$

where the sgn() function is defined as follows:

$$\mathrm{sgn}(x) = \begin{cases} 1, & if\ x \geq 0 \\ 0 & otherwise \end{cases} \qquad (3).$$

In order to increase the robustness of the threshold logic gate, fault tolerances can be added:

$$y = \begin{cases} 1, & if \sum_{i=1}^{n} w_i x \ \geq T + \Delta_1 \\ 0, & if \sum_{i=1}^{n} w_i x_i < T - \Delta_2 \end{cases}, \qquad (4)$$

where parameters Δ_1 and Δ_2 are tolerances that ensure stability with respect to technology and temperature variations which can violate functionality of the system.

A device which implements this theoretical model must compute the linear weighted sum of the binary inputs, store the threshold value and compare the weighted sum to this threshold. TL can be programmed to realize many distinct

[1]Boyan P. Lyubenov is Ph.D. student at Faculty of Electronic Engineering and Technologies, Technical University – Sofia, 1000 Sofia, Bulgaria

[2] Emil D. Manolov is with the Faculty of Electronic Engineering and Technologies, Technical University – Sofia, 1000 Sofia, Bulgaria, E-mail: edm@tu-sofia.bg

Boolean functions by adjusting the threshold T and/or the weights w_i. For example, an n-input TL gate with $T=n$ will implement n-input AND gate, but only by setting $T=n/2$, the gate will compute a majority function. This versatility means that TL offers significantly increased computational capabilities over the conventional AND/OR/NOT logic. Moreover, the reduced area, increased speed and larger number of input variables are part of advantages of TL over conventional Boolean logic.

III. IMPLEMENTATION

Standard logic functions can be easily implemented using threshold logic gates instead of Boolean. A TL gate can directly replace every conventional logic gate. If consider 2-input threshold gate from Fig. 1 and set the weights equal to '1', we can control the property of that gate only by varying the threshold value.

For instance, to obtain logical operation "AND" the gate have to be trained for the input pairs (x_1 x_2) and the output respond y so that (0 0):0, (1 0):0, (0 1):0, (1 1):1. Analysis of Eq. (1) in case of $w_1=w_2=1$ show that if $1<T\leq 2$ the threshold gate is AND. Then if the both inputs are '1' the weighted sum is equal or greater than the threshold T.

The similar approach can be used to implement logical function "OR". In this case the inputs signals and output response are (0 0):0, (1 0):1, (0 1):1, (1 1):1. Analysis of Eq. (1) in case of $w_1=w_2=1$ show that if $0<T\leq 1$ the threshold gate is OR. The results clearly prove that only by changing the threshold value T, AND-gate can be transformed to OR-gate and vice versa. Graphical explanation of the possibility for threshold adjustment is shown in Fig. 2. Threshold value T changes the position of the line that separates the plane into two regions.

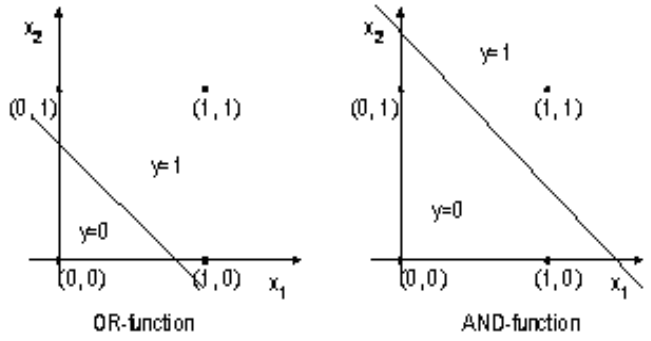

Fig. 1. AND/OR function representation

To implement and simulate AND/OR threshold gates Anadigm Desiger2 program is used. The practical verification is carried out using Anadigm Evaluation board. The circuit is shown on Fig. 2: the summation is done by SumDiff block (Σ) and the comparison - by Comparator. Input levels for x_1 and x_2 are 0V and 1V respectively and the threshold T changes from 0V to 2V. The circuit simulation and verification confirm the transformation of OR-gate to AND-gate and vice versa only by changing the threshold level T (Fig. 3).

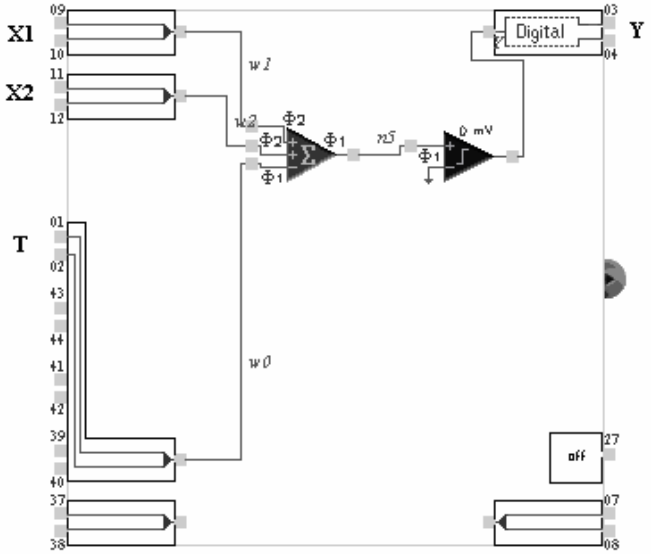

Fig. 2. FPAA implementation of AND/OR gate

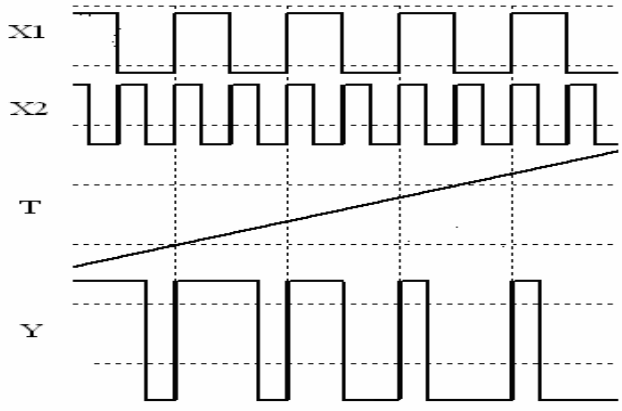

Fig. 3. Simulation of AND/OR function

Implementation of NAND/NOR gate also is easy task – now the basic gate output Eq. (1) is changed into Eq. (5):

$$y=\begin{cases}0, & if \sum_{i=1}^{n} w_i x_i \geq T \\ 1 & otherwise\end{cases}. \qquad (5)$$

In this case, the regions, where y=1 and y=0, are swapped in comparison with AND/OR function. FPAA implementation is identical to the shown on Fig. 2. Only the Comparator block is changed from non-inverting to inverting mode.

In the case of 3-input logic gate, if $2<T\leq 3$ - the threshold gate is AND, if $0<T\leq 1$ - the threshold gate is OR. Increasing the number of the inputs goes not affect the speed of the gate – this is a very important property that attracts designer's attention to threshold logic. The 3-input gate is presented on Fig. 4. The summing element is 4-input SumDiff (Σ), the input levels for x_1, x_2 and x_3 are 0V and 1V respectively, and the threshold T changes from 0V to 3V. The simulations are shown in Fig. 5. Threshold curve is not presented on the picture because of the fact that only 4 cursors are available in Anadigm Desiger2 software [9]. The gate changes its function according to the value of threshold T.

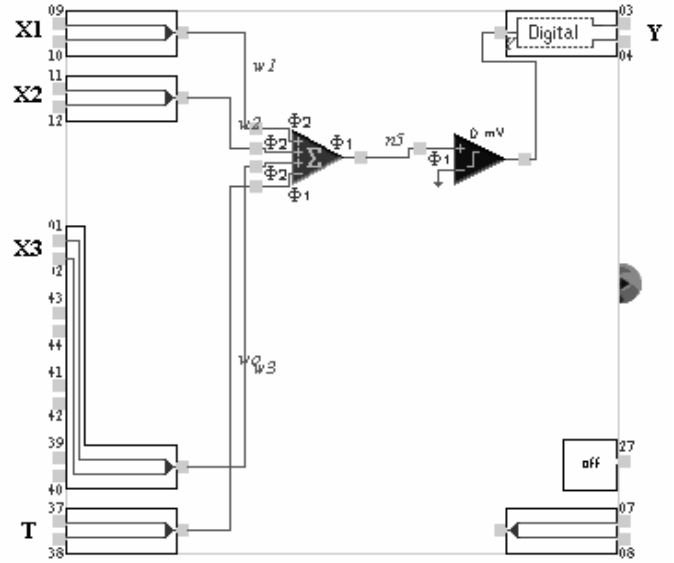

Fig. 4. FPAA implementation of 3-input AND/OR gate

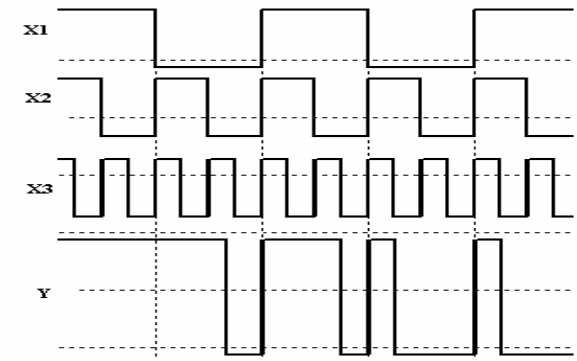

Fig. 5. Simulation of 3-input AND/OR gate

Let consider more complex network such as the logical operation "EXCLUSIVE-OR". The creation of XOR gate is a bit complex task. In this case two-level threshold logic is needed. The logic equation of XOR gate is $y = x_1\bar{x}_2 + \bar{x}_1 x_2$.

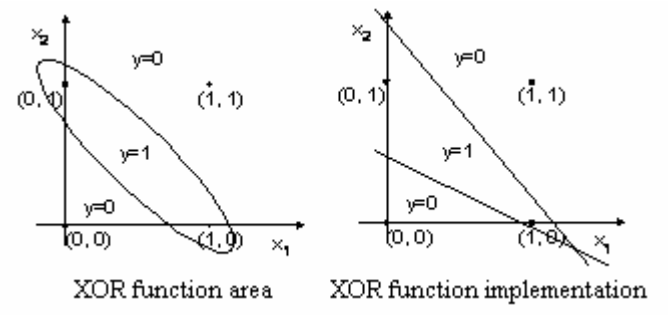

Fig. 6. XOR function representation and implementation

There are several approaches to implement such gate. Graphical explanation is shown in Fig. 6. The left picture shows the area of representation of XOR-function. The right picture present the implementation of the function using AND and OR logical components. The goal is to build a system that performs summation of the outputs of an AND-gate and OR -gate (Fig. 6). The function can be implemented by using two-layer network (Fig. 7). The presented network have to be trained to obtain the following states $(0\,0):0$, $(1\,0):1$, $(0\,1):1$, $(1\,1):0$. Setting $w_{11}=w_{12}=w_{21}=w_{22}=1$, the system have to be solved for T_1, T_2, T_3 and w_{Y1}, w_{Y2} as well. $Y1$ and $Y2$ are the outputs of the hidden layer. One possible solution is to set $0<T_2\leq 1$, $0<T_3\leq 1$ and $1<T_1\leq 2$ and $w_{Y1}= -1$, $w_{Y2}=1$.

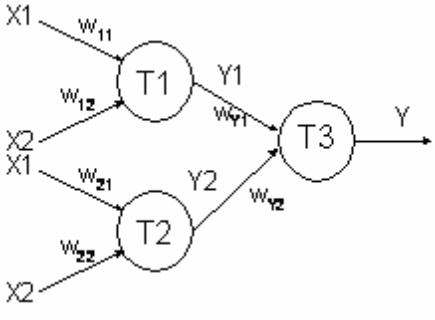

Fig. 7. 2-input 2-layer network

Another possible way to implement a XOR-gate is shown on Fig. 8. In this case: $1<T_1\leq 2$, $0<T_2\leq 1$, $w_{11}=w_{12}=w_{21}=w_{22}=1$, and $w_{Y1}= -2$.

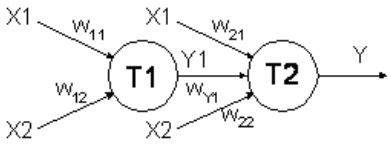

Fig. 8. Another 2-input 2-layer network

The circuit shown in Fig. 8 is more compact and was implemented using FPAA (Fig. 9).

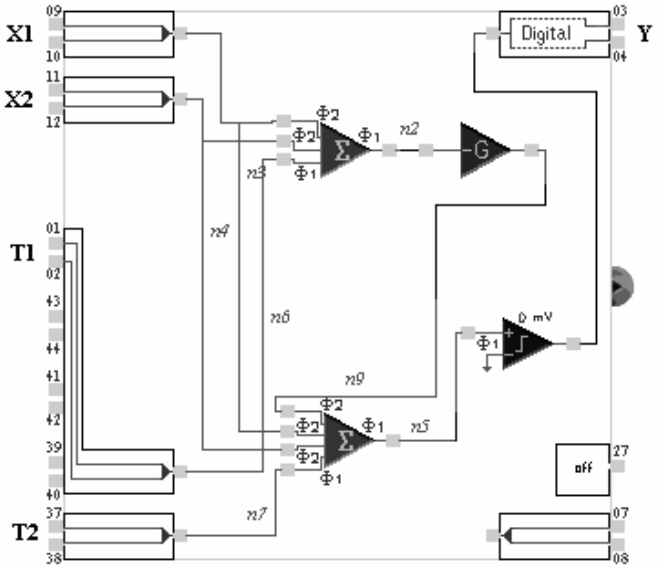

Fig. 9. FPAA implementation of 2-input XOR gate

The gate T1 is represented by 3-input SumDiff (Σ) and the gate T2 by 4-input SumDiff (Σ). GainInv (-G) cell represents the weight w_{Y1}=-2 as was derived above. The thresholds of the gates are set T_1=1.5V and T_2=0.5V. At these conditions the simulations (Fig. 10) clearly show that the presented FPAA implementation is XOR-gate.

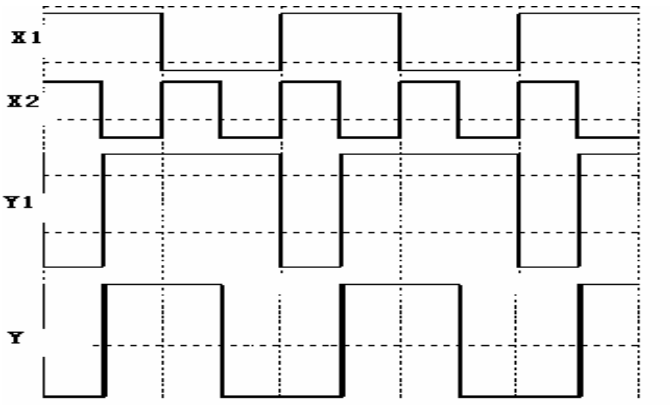

Fig. 10. Simulation of 2-input XOR function

Another experiment is to build more complex functions. There are two basic approaches: either to replace every logic function with the appropriate threshold logic gate or to train a special gate to satisfy the wanted function. Two examples are presented in order to demonstrate these basic options.

Let the wanted function is $y = x_1x_2 + x_3$. Using the first method every operation have to be implemented by a single TL-gate: firstly $y_1 = x_1x_2$ and then $y = y_1 + x_3$. That means two gates and two threshold values. The way to build 'AND' and 'OR' gates was presented above.

The second approach is to use 3-input TL gate. For the input vectors (x_1 x_2 x_3) the following response y is needed: (0 0 0):0, (0 0 1):1, (0 1 0):0, (0 1 1):1, (1 0 0):0, (1 0 1):1, (1 1 0):1, (1 1 1):1. Analysis of Eq. (1) in the case of w_1=w_2, gives the following:

$$T>0, \qquad 0.5T< w_1=w_2<T, \qquad w_3>T \qquad (6)$$

Every set of values that satisfy the system (6) represents the given logic function. This approach reduces the final gate number and the complexity of the entire system.

FPAA implementation of the discussed function is very easy. As summing element is used 4-input SumDiff (Σ) and the weights are set as coefficients within that block (Fig. 4). One possible solution of (6) to achieve $y = x_1x_2 + x_3$ is T=1V, w_1=w_2=0.6, w_3=1.2, input levels for x_1, x_2 and x_3 - 0V and 1V respectively for logic '0' and logic '1'. The simulation results are given on Fig. 11.

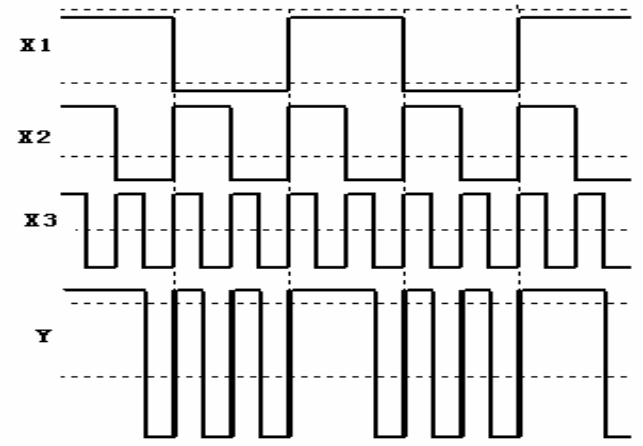

Fig. 11. Simulation of $y=x_1x_2+x_3$ function

Another logical function is $y = (x_1 + x_2)x_3$. Now the input vectors and the output response are (0 0 0):0, (0 0 1):0, (0 1 0):0, (0 1 1):1, (1 0 0):0, (1 0 1):1, (1 1 0):0, (1 1 1):1. The analysis of Eq. (1), for w_1=w_2, gives:

$$T>0, \qquad 0< w_1=w_2<0.5T, \qquad T>w_3>T-w_1 \qquad (7)$$

The function was implemented using FPAA in the same way as the previous. The conditions are: T=1V, w_1=w_2=0.4, w_3=0.8, input levels for x_1, x_2 and x_3 - 0V and 1V respectively for logic '0' and logic '1'. The results from simulation confirm the correctness of the presented deductions.

More complex combinatory functions such multiplexers, demultiplexers and many other can be implemented using the approach and basic threshold logic gates proposed above.

IV. CONCLUSIONS

The paper presents an approach for prototyping and examination of basic threshold logic functions using FPAA. To this aim, the functional model of the threshold logic gate is discussed and some approaches in building logic functions are presented. The results from implementation of different FPAA variants of threshold logic structures are described. The use of the FPAA ensures possibilities for simple programming and dynamic reconfiguration of different values of the weights on the inputs as well as a flexible realization of different logic functions. The results clearly present the practical use of the discussed approach and could find application for fast prototyping of threshold logic based systems with possibilities for flexible real-time programming and reconfiguration of the parameters and functions.

REFERENCES

[1] McCulloch, W.S., Pitts, W., "A logical calculus of the ideas implement in nervous activity," Bull. Math. Biophysiol., Vol. 5, 1943, pp. 15-33.

[2] Rosenblatt, F., "The perceptron a probabilistic model for information storage and organization," Brain Psych. Rev., Vol. 62, 1958, pp. 368-408.

[3] Huertas, J.L., Sanchez-Solano, S., Baturone, I., Barriga, A., "Integrated circuit implementation of fuzzy controllers" IEEE JSSC, Vol. 31, NO. 7, July 1996, pp. 1051-1058.

[4] Espejo, S., Domingues-Castro, R., Rodrigues-Vazques, A., "A 16x16 cellular neural network chip for connected component detection", 1993

[5] Leong, P.H.W., Jabri, M.A., "A VLSI neural network for morphology classification", International Joint Conference on Neural Networks, IJCNN, 1992, Volume: 2, pp. 678-683.

[6] Berenson, D., N. Estevez, H. Lipson, "Hardware Evolution of Analog Circuits for In-situ Robotic Fault-Recovery," 2005 NASA/DoD Conference on Evolvable Hardware (EH'05), 2005 , pp. 12-19.

[7] Manolov E.D., B.P. Lyubenov, Design and investigation of two-parameter space classification circuits using FPAA. Proceedings of the 14th International Scientific and Applied Science Conference Electronics ET'2005, book 5, pp.99-104.

[8] Manolov, E.D., "Research and Educational Experiments with FPAA", 12th International Conference Mixed Design of Integrated Circuits and Systems MIXDES'2005. Krakow, Poland, vol.2. pp. 975-980.

[9] Anadigm Inc. Technical Documentation. www.anadigm.com