ISSN 0002-306X. Изв. НАН РА и ГИУА. Сер. ТН. 2014. Т. LXVII, № 1.

UDC 621.382.13 **RADIOELECTRONICS**

O.H. PETROSYAN, V.P. GRIGORYANTS, G.A. KARAPETYAN, A.R. GRIGORYAN

DEVELOPING AND INVESTIGATING A METHOD FOR DIGITAL FORMAL NEURON DESIGN

A method for digital formal neurons synthesis is developed and implemented by means of C++ programing language. The method developed, compared with the simplex method of formal neurons synthesis, is characterized by low complexity. A digital formal neuron analysis is performed. An optimality criterion is proposed, which makes it possible to give some evaluation of the synthesized formal neuron.

Keywords: digital formal neuron, neural network, software of automatic synthesis, simulation.

Introduction. An artificial neural network (ANN) is a system that is based on operations of biological neural networks, and hence can be defined as an emulation of biological neural systems. ANN's are at the forefront of computational systems designed to produce, or at least mimic, intelligent behavior [1,2]. ANN consists of millions of formal neurons connected to each other, the model of the biological neuron. We will investigate a formal neuron with "sign" activation function [3-5]. The first model of DFN was proposed by McCulloch and Pitts. This model implements the function of threshold logic units (TLU) [6].

$$y = sign(\sum_{i=1}^{n} \omega_i x_i - \Theta). \quad (1)$$

But, it is known that with one TLU, it is impossible to implement all Boolean functions, because of linear inseparability [3-5,7,8]. Then DFN was developed and input interactions were added, which helps DFN to implement all Boolean functions [4,5]. In a general case DFN has a view shown in Fig. 1.

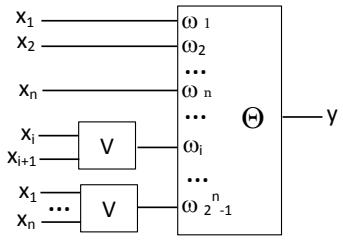

Fig. 1. Graphical view of DFN

where, instead of block "V" it could be any of Boolean functions. For some certainty we will discuss DFN with "AND" input interaction and the results will be generalized for the remaining input interactions.

A method for the DFN design. In the known methods of DFN synthesis a lot of parameters are required [4,5]. For example a DFN specified by Venn's threshold diagram requires sums of weights for all possible input combinations. It means that DFN with n inputs will require 2^n parameters. It may cause some technical difficulties. Another question also arises, how should weights and threshold be calculated to implement the necessary Boolean function.

In this work, for developing a method of DFN synthesis, the following has been taken into account. From a technical point of view, it is more important and easier to give the amount of DFN inputs and the function which should implement the synthesized DFN.

To synthesize a DFN it is necessary to construct a system of inequalities for all possible combinations of inputs, and solve it according to the given Boolean function. In Table 1, all inequalities for each input combination are shown, and the inequality which is responsible for the appropriate digit y_i of the given function y.

Table 1

№	x_n	…	x_2	x_1	sign(ω,x)	y
1	0	…	0	0	$0 \geq \Theta$	y_0
2	0	…	0	1	$\omega_1 \geq \Theta$	y_1
3	0	…	1	0	$\omega_2 \geq \Theta$	y_2
4	0	…	1	1	$\omega_1+\omega_2 \geq \Theta$	y_3
…	…	…	…	…	…	…
2^n	1	1	1	1	$\omega_1+\omega_2+\ldots+\omega_{2^n-1} \geq \Theta$	y_{2^n-1}

If the appropriate digit y_i of the given function y is equal to 0, the inequality is false, and vice versa if the appropriate digit y_i of the given function y is equal to 1, the inequality is true. For example if y_3=0, $\omega_1+\omega_2 < \Theta$, and if y_3=1, $\omega_1+\omega_2 \geq \Theta$.

Now let's understand the principle of the developed method of DFN synthesis. For simplicity, let's observe DFN with 3 inputs and construct a characteristic inequality system (2).

$$\begin{cases} 0 \geq \Theta, \\ \omega_1 \geq \Theta, \\ \omega_2 \geq \Theta, \\ \omega_1 + \omega_2 + \omega_4 \geq \Theta, \\ \omega_3 \geq \Theta, \\ \omega_1 + \omega_3 + \omega_5 \geq \Theta, \\ \omega_2 + \omega_3 + \omega_6 \geq \Theta, \\ \omega_1 + \omega_2 + \omega_3 + \omega_4 + \omega_5 + \omega_6 + \omega_7 \geq \Theta. \end{cases} \quad (2)$$

Let the given function be "01100000". Each "0" or "1" digit is the function output for an appropriate line of the system (2). The first "0" digit shows, that the first inequality is false. It is possible only for positive thresholds. For optimality the software will set the threshold to be equal to possible minimal discrete positive value,

i.e. Θ=1. The first "1" digit shows that the second inequality is true. It is possible only for the weights equal to or greater than 1. For optimality, the software will set the weight more than the threshold by the possible discrete value, i.e. $\omega_1=1$. Continuing in such a way, we can calculate all weights and thresholds for the given function. This method is applicable for any amount of inputs and for implementing any Boolean function by the synthesized DFN.

The algorithm described above is implemented by C++ programming language and is given below.

```
1.  int neuron::solve_neuron(unsigned long a_function)
2.   {
3.      if(a_function>functions)
4.     {
5.         cout<<"Wrong function\n";
6.         return 1;
7.  }
8.  //solving code here
9.  theta=0;
10. for(int i=0;i<n_synapses;i++)
11.     omega_vector[i]=0;
12. bitset<512> f(a_function);
13.
14. if(f[0]==0)
15.     theta=1;
16. int temp=0;
17. for(int i=1;i<=n_synapses;++i)
18. {
19.     for(int j=0;j<n_synapses;j++)
20.        //find last 1(unknown)
21.        if(omega[i][j]==1)
22.           temp=j;//temp will remember the last 1 which is unknown always
23. //now we will count the sum of omegas befor this unknown
24. int sum=0;
25. for(int e=0;e<temp;e++)
26.     sum+=omega[i][e]*omega_vector[e];
27. //now we have omega(temp)>=theta-sum, so we will do omega(temp)=theta-sum for F=1
28. //and omega=theta-sum-1 for F=0(so condition will not met and f will be 0)
29. if(f[i]==1)
30.   omega_vector[temp]=theta-sum;
31.      else
32.         omega_vector[temp]=theta-sum-1;
33.         }
34.         return 0;
35. }
```

As you can see from line 17, the algorithm solves the system by one cycle which repeats (n_synapses+1) times, where n_synapses is the amount of synapses (weights). Let n=n_synapses+1, where n is the amount of variables, then the complexity of the algorithm will be O(n). Comparing the developed method with DFN synthesis by the simplex method [4,5,9], the complexity of which in the average case is $O(n^3)$, and in the worst case is $O(n^2 2^n)$ [10], we gain a lot of time. For visual comparison in Fig. 2, you can find the complexity difference graphics between the developed algorithm and the simplex one in the logarithmic scale.

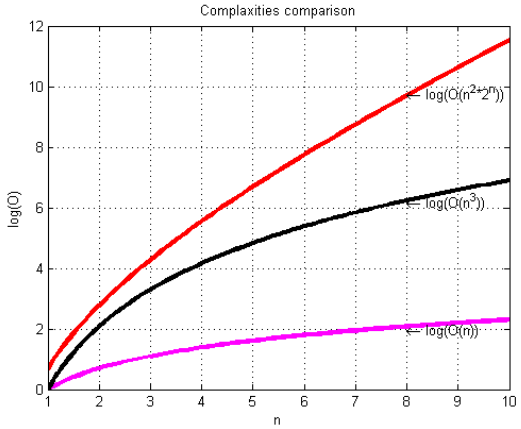

Fig. 2. Complexity comparison between the developed algorithm and the simplex. log(O($n^2 2^n$)) – the worst case of the simplex method, log(O(n^3) – the average case of the simplex method, log(O(n)) – the developed method

Optimality criterion. First of all, we are interested in DFN implementation in hardware. Thus, from a technical point of view, it is important to have as minimal deviation as possible of the DFN parameters. Besides, to evaluate the efficiency of the developed algorithm, the following optimality criterion is proposed:

$$R = \frac{\max \omega_i - \min \omega_j}{2^n},$$

where $1 \le i, j \le 2^n - 1$, $\max \omega_i$, $\min \omega_j$ are correspondingly the maximum and minimum weights for all possible Boolean functions. It is obvious that the more R is near to 1, the lower is the deviation of DFN parameters, because DFN with discrete valued weights should have maximum changing range of weights to be equal to not lower than 2^n to make possible to implement all Boolean functions. So if R<1, there is at least one function that cannot be implemented by the synthesized DFN. If R>1, we have a lack of deviation, and it can be reduced to be equal to 1. So, summarizing, to be sure that we have calculated the weights and the threshold of DFN optimally, it is necessary and sufficient to have R=1. Now let's prove that the developed algorithm

ensures the optimal criterion to be equal to 1. If the first digit of the given function y_0=0, as it has already been mentioned, the software will set Θ to be equal to 1 (See Table 1). In the other case if y_0=1, the software will set Θ to be equal to 0. Thus, we can write the following $\Theta = \overline{y_0}$. In the same way, we can write for the remaining lines of Table 1: $\omega_1 = \overline{y_0} - \overline{y_1}$, $\omega_2 = \overline{y_0} - \overline{y_2}$, $\omega_3 = -\overline{y_0} + \overline{y_1} + \overline{y_2} - \overline{y_3}$, etc. So after the calculations, we will have Table 2.

Table 2

№	x_n	…	x_2	x_1	sign(ω,x)	y	DFN parameters calculation
1	0	…	0	0	$0 \geq \Theta$	y_0	$\Theta = \overline{y_0}$
2	0	…	0	1	$\omega_1 \geq \Theta$	y_1	$\omega_1 = \overline{y_0} - \overline{y_1}$
3	0	…	1	0	$\omega_2 \geq \Theta$	y_2	$\omega_2 = \overline{y_0} - \overline{y_2}$
4	0	…	1	1	$\omega_1+\omega_2+\omega_3 \geq \Theta$	y_3	$\omega_3 = -\overline{y_0} + \overline{y_1} + \overline{y_2} - \overline{y_3}$
…	…	…	…	…	…	…	…
2^n	1	1	1	1	$\omega_1+\omega_2+\ldots+\omega_{2^n-1} \geq \Theta$	y_{2^n-1}	$\omega_{2^n-1} = \Theta - \sum_{i=1}^{2^n-2} \omega_i - \overline{y_{2^n-1}}$

Here, by adding and subtracting, we should understand the arithmetical operation and not the logical. It is obvious that the weight which has maximal members is ω_{2^n-1}.

$$\omega_{2^n-1} = \Theta - \sum_{i=1}^{2^n-2} \omega_i - \overline{y_{2^n-1}}. \quad (3)$$

Thus, it will specify the maximum and minimum values among all weights for all Boolean functions. Let's discuss the last weight ω_{2^n-1} of DFN with 2, 3, 4, ..., n inputs.

$$n = 2, \omega_3 = -\overline{y_0} + \overline{y_1} + \overline{y_2} - \overline{y_3}, \quad (4)$$

$$n = 3, \omega_7 = \overline{y_0} - \overline{y_1} - \overline{y_2} - \overline{y_3} + \overline{y_4} + \overline{y_5} + \overline{y_6} - \overline{y_7},$$

$$n = 4, \omega_{15} = -\overline{y_0} + \overline{y_1} + \overline{y_2} + \overline{y_3} + \overline{y_4} - \overline{y_5} - \overline{y_6} - \overline{y_7} - \overline{y_8} - \overline{y_9} - \overline{y_{10}} + \overline{y_{11}} + \overline{y_{12}} + \overline{y_{13}} + \overline{y_{14}} - \overline{y_{15}}.$$

Continuing in the same way, we can write the following:

$$\omega_{2^n-1} = (-1)^{n+1}\left(\overline{y_0} + \sum_i \overline{y_i}\right) + (-1)^n \sum_j \overline{y_j} - \overline{y_{2^n-1}}.$$

Let the total amount of members for the first sum ($\sum_i \overline{y_i}$) is A1, and for the second one ($\sum_j \overline{y_j}$) be A2, in this case we can write the following:

$$A_1 = C_n^1 + C_n^3 + \cdots + C_n^{2k-1},$$

$$A_2 = C_n^2 + C_n^4 + \cdots + C_n^{2k},$$

where k=1,2,3,… So, taking into account the known equation $2^n = \sum_{i=0}^{n} C_n^i$, we can write the following:

if n is even: $A_1 = 2^{n-1} - 2, A_2 = 2^{n-1}$,
if n is odd: $A_1 = A_2 = 2^{n-1} - 1$.

Now let's calculate $max(\omega_{2^n-1})$ and $min(\omega_{2^n-1})$. For this purpose, let's discuss 2 cases:

1. When n is even: $max(\omega_{2^n-1})$ will be determined if $\overline{y_0} + \sum_i \overline{y_i} = 0, \overline{y_{2^n-1}} = 0$ ($y_i \in \{0,1\}, 0 \le i \le 2^n - 1$), as they are negative components and if each member of $\sum_j \overline{y_j}$ is equal to 1: Thus for $max(\omega_{2^n-1})$ we will have:

$$max(\omega_{2^n-1}) = A_2 = 2^{n-1}$$

In the same way, $min(\omega_{2^n-1})$ will be determined if $\sum_j \overline{y_j} = 0, \overline{y_0} = 1$, $\overline{y_{2^n-1}} = 1$ and if each member of $\sum_i \overline{y_i}$ is equal to 1:

$$min(\omega_{2^n-1}) = -(A_1 + 2) = -2^{n-1}$$

$$R = \frac{\max \omega_{2^n-1} - \min \omega_{2^n-1}}{2^n} = \frac{2^{n-1} - (-2^{n-1})}{2^n} = 1.$$

2. When n is odd: $max(\omega_{2^n-1})$ will be determined if $\sum_j \overline{y_j} = 0, \overline{y_{2^n-1}} = 0, \overline{y_0} = 1$ and if each member of $\sum_i \overline{y_i}$ is equal to 1. Thus we will have

$$max(\omega_{2^n-1}) = A_1 + 1 = 2^{n-1},$$

$min(\omega_{2^n-1})$ will be determined if $\overline{y_0} + \sum_i \overline{y_i} = 0, \overline{y_{2^n-1}} = 1$ and if each member of $\sum_j \overline{y_j}$ is equal to 1. Thus we will have:

$$min(\omega_{2^n-1}) = -A_2 - 1 = -2^{n-1}$$

$$R = \frac{\max \omega_{2^n-1} - \min \omega_{2^n-1}}{2^n} = \frac{2^{n-1} - (-2^{n-1})}{2^n} = 1.$$

As you can see, the developed algorithm allows to calculate the weights and the threshold of the DFN optimally.

It is also possible to ensure the efficiency of the developed algorithm by the following considerations. Let's discuss DFN with 2 inputs n=2. As you can see from equation (4), the range of ω_3 is $\omega_3 \in [-2:2]$, thus R=(2-(-2))/4=1. It can be shown that DFN with 3 and more inputs can be constructed with multiple DFNs with 2 inputs. But as weights and thresholds of each DFN with 2 inputs are calculated efficiently, we will get efficient weights and thresholds for the whole system. Therefore, by making a reverse operation of coming back from mutually connected multiple DFNs to the DFN with 3 and more inputs; we will get optimally calculated weights and thresholds for the DFN with 3 and more inputs.

Experimental results. In Fig. 3 the results of the developed software for the DFN with 3 inputs are shown which implement 6 (01100000) function. As you can see, the synthesized DFN has the following parameters:

Number of inputs: 3;

The total amount of weights (synapses): 7;

The total amount of all possible Boolean functions which can be implemented by the DFN with 3 inputs: 256;

Weights: {1; 1; 0; -2; -1; -1; 2}

Threshold: Θ=1

Implementing function: 6 (01100000)

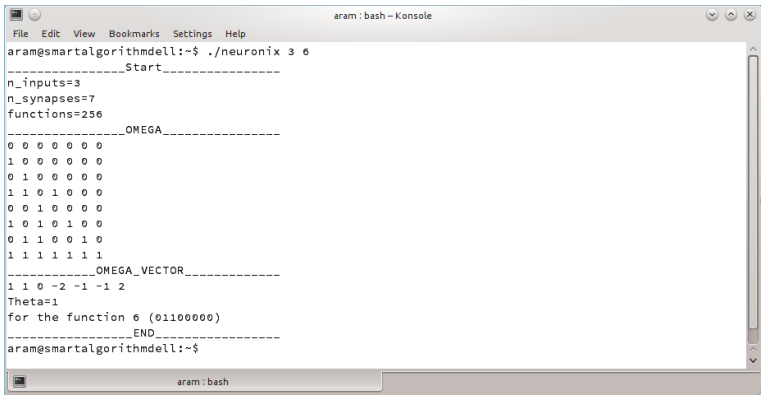

Fig. 3. The example of synthesized DFN by the developed software

The developed software also has a peculiarity to generate the description file of the synthesized DFN by the Verilog-A modeling language. This Verilog-A description file can be used in further DFN simulations. Below, you can find the Verilog-A description file of the synthesized DFN.

```
/*******************************************************
Verilog-A file of a neuron
Created by Neuronix v 0.0.1
*******************************************************/
`include "constants.vams"
`include "disciplines.vams"
module va_neuron(Y_OUT,X_INPUT,W_WEIGHT,T_TRESHHOLD,vp,gd);
output Y_OUT; //Y_OUT is output
electrical Y_OUT;
input [2:0] X_INPUT; //X_INPUT is for inputs
electrical [2:0] X_INPUT;
input [6:0] W_WEIGHT; //W_WEIGHT is for weights of synapses
electrical [6:0] W_WEIGHT;
input T_TRESHHOLD; //T_TRESHHOLD is the treshhold of the neuron
electrical T_TRESHHOLD;
real sum=0; //initializing sum for weights
analog begin
```

```
sum=V(W_WEIGHT[0])*V(X_INPUT[0])
+V(W_WEIGHT[1])*V(X_INPUT[1])
+V(W_WEIGHT[2])*V(X_INPUT[2])
+V(W_WEIGHT[3])*(V(X_INPUT[0])&&V(X_INPUT[1]))
+V(W_WEIGHT[4])*(V(X_INPUT[0])&&V(X_INPUT[2]))
+V(W_WEIGHT[5])*(V(X_INPUT[1])&&V(X_INPUT[2]))
+V(W_WEIGHT[6])*(V(X_INPUT[0])&&V(X_INPUT[1])&&V(X_INPUT[2]));
if(sum>=V(T_TRESHHOLD) begin sum = V(vp); end //sign function check
else begin sum = V(gd); end
V(Y_OUT) <+ transition (sum); // making output to Y_OUT
end
endmodule
```

Where “Y_OUT” is the output pin, “X_INPUT” is the input pin, “W_WEIGHT” is the weights, “T_THRESHOLD” is the threshold of the synthesized DFN. “vp” is the power supply and “gd” is the ground. The variable “sum” calculates the weighed sum of the input, and if it is more than the threshold, Y_OUT=1, otherwise Y_OUT=0.

The synthesized Verilog-A description file was used in Synopsys HSPICE simulation, the timing diagrams of which are shown in Fig. 4. As you can see, the simulation was carried out for all input combinations and the synthesized DFN implements the given function 6 (01100000).

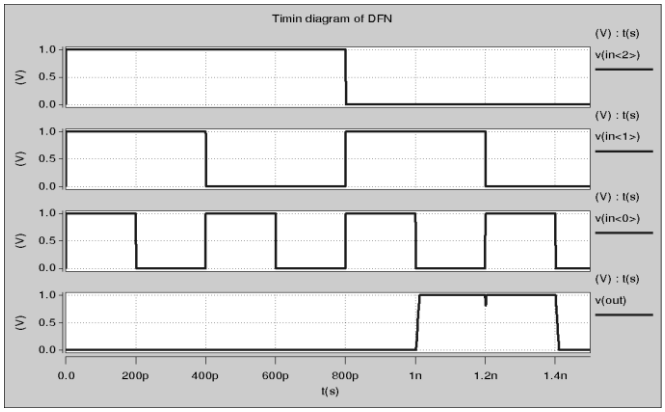

Fig. 4. Timing diagram of DFN with 3 inputs and which implements 01100000 function

Conclusion. The developed method, unlike the existing ones, has very low complexity. The developed method satisfies the optimality criterion, which has been proposed by the author, and which allows to calculate the weights and the threshold efficiently from the technical point of view. The developed method was implemented by the software written in C++ programming language. The developed software also generates Verilog-A description file of the synthesized DFN which can be used in HSPICE modeling software for further simulations. The workability of the developed software has been checked and verified by many tests, one of which is presented in this paper.

References

1. http://www.dhtusa.com/media/NeuralNetworkIntro.pdf
2. From Neuron to Brain, Fifth edition / **G. John Nicholls, A. Robert Martin, Paul A. Fuchs, et al.** - November 7, 2011.-621p.
3. **Haykin Simon.** Neural Networks. A Comprehensive Foundation. Second Edition.- Published by Pearson Education.- July 16, 1998.- 823 p.
4. **Мкртчян С.О., Мкртчян А.С.** Основы цифровой нейроинформатики: Учебное пособие. – Ереван: Изд-во ГИУА “Чартарагет”, 2007. – 387 с.
5. **Мкртчян С.О.** Проектирование логических устройств ЭВМ на нейронных элементах. – М.: Энергия, 1977. – 200 с.
6. **Warren S. McCulloch and Walter Pitts.** A logical calculus of the ideas immanent nervous activity // Bulletin of Mathematical Biophysics.- 1943.- Vol. 5.- P.115-133.
7. **Rojas R.** Neural Networks. - Springer-Verlag, Berlin, 1996.
8. http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html
9. **Scott R. J.** Construction of Neuron Model//Ire Trans on Bio-Med. Electronics. –1961.- Vol. 8, № 5.-P.198-202.
10. http://www.iip.ist.i.kyoto-u.ac.jp/member/cuturi/Teaching/ORF522/lec8v2.pdf

SEUA (POLYTECHNIC). The material is received 10.12.2013.

Հ.Հ. ՊԵՏՐՈՍՅԱՆ, Վ.Պ. ԳՐԻԳՈՐՅԱՆՑ, Գ.Ա. ԿԱՐԱՊԵՏՅԱՆ, Ա.Ռ. ԳՐԻԳՈՐՅԱՆ

ԹՎԱՅԻՆ ՖՈՐՄԱԼ ՆԵՅՐՈՆՆԵՐԻ ՆԱԽԱԳԾՄԱՆ ՄԵԹՈԴԻ ՄՇԱԿՈՒՄԸ ԵՎ ՀԵՏԱԶՈՏՈՒՄԸ

Մշակվել է թվային ֆորմալ նեյրոնների նախագծման մեթոդ, որն իրականացվել է C++ ծրագրավորման լեզվի միջոցով: Մշակված մեթոդը, ի տարբերություն ֆորմալ նեյրոնների սինթեզման սիմպլեքս մեթոդի, բնութագրվում է նվազ բարդությամբ: Կատարվել է թվային ֆորմալ նեյրոնների վերլուծություն: Առաջարկվել է օպտիմալության չափանիշ, որով կարելի է գնահատել սինթեզված ֆորմալ նեյրոնները:

Առանցքային բառեր. թվային ֆորմալ նեյրոն, նեյրոնային ցանց, ավտոմատ նախագծման ծրագիր, սիմուլյացում:

О.А. ПЕТРОСЯН, В.П. ГРИГОРЯНЦ, Г.А. КАРАПЕТЯН, А.Р. ГРИГОРЯН

РАЗРАБОТКА И ИССЛЕДОВАНИЕ МЕТОДА ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ ФОРМАЛЬНЫХ НЕЙРОНОВ

Разработан метод проектирования цифровых формальных нейронов, который реализован с помощью языка программирования C++. Разработанный метод, по сравнению с симплекс методом синтезирования формальных нейронов, характеризуется малой сложностью. Проведен анализ цифровых формальных нейронов. Предложен критерий оптимальности, с помощью которого можно оценить синтезированные формальные нейроны.

Ключевые слова: цифровой формальный нейрон, нейронная сеть, программа автоматического проектирования, симуляция.