Simplification and Implementation of Boolean Functions

Hazem M. El-Bakry*, **Ahmed Atwan****

** Department of Information Systems, Faculty of Computer Science and Information Systems, Mansoura University, Mansoura, EGYPT*
tel: +2 050 2349 340, fax: +2 050 2221 442
e-mail: helbakry20@yahoo.com
*** Department of Information Technology, Faculty of Computer Science and Information Systems, Mansoura University, Mansoura, EGYPT*

Submitted: 26/12/2009
Accepted: 10/01/2010
Appeared: 16/01/2010

Abstract— In previous work (El-bakry, H. M., Mastorakis N., (2009)), a fast systematic method for minimization of the Boolean functions was presented. Such method is a simple because there is no need for any visual representation such as Karnough map or arrangement technique such as Tabulation method. Furthermore, it is suitable for boolean function with large number of variables (more than 4 variable). Moreover, it is very simple to understand and use. In this paper, the simplified functions are implemented with minimum amount of components. A powerful solution for realization of more complex functions is given. This is done by using modular neural nets (MNNs) that divide the input space into several homogenous regions. Such approach is applied to implement XOR functions, 16 logic function on one bit level, and 2-bit digital multiplier. Compared to previous non- modular designs, a clear reduction in the order of computations and hardware requirements is achieved.

Keywords: Boolean Functions, Simplification, Implementation, MNNs

1. INTRODUCTION

The simplification of Boolean functions is mainly used to reduce the number of gates in a logic circuit. Less number of gates means less power consumption, sometimes the circuit works faster and also when number of gates is reduced, cost also comes down (Marcovitz, A. B., (2007), Mano, M. M., and Ciletti, M. D., (2003) & Mano, M. M., (1984)). There are many ways to simplify a logic design, such as algebraic simplification, Karnough maps, Tabulation Method and Diagrammatic technique using 'Venn-like diagram' some of them are discussed in detail in this introduction (Marcovitz, A. B., (2005), Arntson, A. E., (2005), Mano, M. M., Ciletti, M. D., (2003), & Mano, M. M., (1984)).

In this paper, a new fast systematic method for minimization of the Boolean function is introduced. Such method is a very simple because there is no need to any visual representation such as Karnough map or arrangement technique such as Tabulation method and very easy for programming. This method is very suitable for high variables (more than 4 variable) boolean function, and very simple for students (Mano, M. M., and Ciletti, M. D., (2003) & Mano, M. M., (1984)). Furthermore, neural networks are used to implement Boolean functions because they can recognize patterns even with noise, distortion or deformation in shape. This is very important especially in communication applications.

1.1 Boolean Functions

A Boolean function is an expression consisting of binary variable operators OR, AND, the operator NOT, parentheses, and an equal sign. For a given value of these variables, the function can be either 0 or 1. Consider, for example, the following Boolean function (Atwan, A. (2006), Marcovitz, A. B., (2007) & Mano, M. M., Ciletti, M. D., (2003)): $F=X+Y'Z$, F equal *1*, when $X=1$ or Y=0, while $Z=1$.

1. Rules of Boolean Algebra:

The standard rules of Boolean algebra which reproduce for simplicity are introduced in table 1:

Table 1: Rules of Boolean algebra

X + X = X	X . X = X
X + 0 = X	X . 1 = X
X + 1 = 1	X . 0 = 0
X + X' = 1	X . X' = 0

2. Canonical and Standard Form (Minterms)

A binary variable may come into view either in its normal form, *X*, or in its complement form, *X'*. Now consider two binary variables X and Y combined with AND operations.

Since each variable may appear in each form, there are four possible combinations, namely, *XY*, *XY'*, *X'Y*, and *X'Y'*. Each of these four terms is called a Minterm or a standard product. In a similar way, *N* variables can be combined to form 2^n Minterms. The 2^n different Minterms may be determined by a method similar to the one shown in Table 1 which shows the case of 3 variables (Marcovitz, A. B., (2007), Arntson, A. E., (2005), Vahid, F., (2006) & Hayes, J. P. (1993)).

Table 2. Combination of Minterms for 3 variables

Decimal Form	X	Y	Z	Term	Designation
0	0	0	0	X'Y'Z'	m0
1	0	0	1	X'Y'Z	m1
2	0	1	0	X'YZ'	m2
3	0	1	1	X'YZ	m3
4	1	0	0	XY'Z'	m4
5	1	0	1	XY'Z	m5
6	1	1	0	XYZ'	m6
7	1	1	1	XYZ	m7

A Boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variable which produces a 1 in the function and then taking the OR of all those terms. For example, the function F_1 in Table 2 is determined by expressing the combination 001, 100, and 111 as X'Y'Z, XY'Z', and XYZ. Each one of these minterms results in the expression, so F_1 can be expressed as:

$$F_1 = xyz + xyz + xyz = m1+m4+m7$$

It may be more suitable to express the boolean function in the following short notation: $F_1(x,y,z) = \sum(1,4,7)$

Table 3. Representation of F1

X	Y	Z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

1.2 Traditional Methods for Simplification of Boolean Functions

There are many Traditional Methods to simplify a *Boolean Functions*, such as Algebraic Simplification, Karnough Maps and Tabulation Method. This part discusses the frequently used method such as *Karnough Map* and *Tabulation Method*.

1. Map Method (Karnough Map)

Karnough Map is a visual representation diagram of all possible ways a function may be expressed. Map method is introduced by Veich and slightly modified by Karnough .A K-map consists of a grid of squares, each square representing one canonical minterm combination of the variables or their inverse. The map is arranged so that squares representing minterms which differ by only one variable are adjacent both vertically and horizontally. Therefore XY'Z' would be adjacent to X'Y'Z' and would also adjacent to XY'Z and XYZ' (Marcovitz, A. B., (2007), Hayes, J. P. (1993), Arntson, A. E., (2005), & Mano, M. M., (1984)). Example : Simplify the boolean function:

F=W'X'Y'Z + W'X'YZ + W'XY'Z + W'XYZ + WXY'Z'+ WX'Y'Z' or $F(X,Y,Z,W) = \sum(1,3,5,7,8,12)$

Solution:

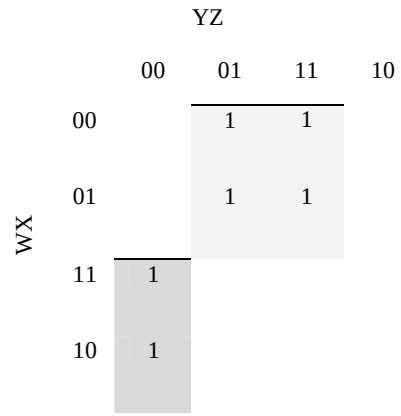

Fig. 1. Karnough Map

F = W'Z + WY'Z'

2. Tabulation Method (QUINE AND MC CLUSKEY)

The Map method of simplification is convenient as long as the number of variable is suitable number. The excessive number of squares prevents a reasonable selection of adjacent squares .The tabulation methods overcomes this difficulty. It is a specific step by step procedure that is guarantied to produce a simplified standard -form expression for the function. the tabular method of simplification consists of two parts. The first is to find by an exhaustive search of all the term that are candidates for inclusion in the simplified function. These terms are called Prime-Implicants .The second operation is to choose among the prime-Implicants those that give an expression with the least number of literals (Mano, M. M., (1984), Floyd, T. L., (2006), Hayes, J. P. (1993), Biswas, N. N., (1984) & Dueck, R., (2004)).

Example: Simplify the following Boolean function by using the tabulation methods: $F(W,X,Y,Z) = \sum(0,1,2,8,10,11,14,15)$

SOLUTON:

Table 4. The Prime Implicants

	a		B		c	
	wxyz		wxyz			wxyz
0	0000	0,1	000-		0,2,8,10	-0-0
		0,2	00-0		0,8,2,10	-0-0
1	0001	0,8	-000			
2	0010				10,11,14,15	1-1-
8	1000	2,10	-010		10,14,11,15	1-1-
		8,10	10-0			
10	1010					
		10,11	101-			
11	1011	10,14	1-10			
14	1110					
		11,15	1-11			
15	1111	14,15	111-			

2. A NEW METHOD FOR SIMPLIFICATION OF BOOLEAN FUNCTIONS

Starting point is offering the main definitions and terminology to be used throughout this method.

- The number of Minterms equals 2^n where n is the number of variables.
- The maximum Minterm to be obtained equals 2^n -1.
- The 1's complement of any binary number turns 0 to 1 and vise versa , for instance 1's complement f 110101 equals 001010 .

Definition 1:
The combination of two Minterms is called "double minimization ".The smaller Minterm is called The base and The other is called The follower.

Definition 2:
The combination of four Minterms is called " quadruple minimization " in which the smaller two Minterms are called the base and the other two Minterms are the followers .

Theorem 1:
If X is the number which represent any Minterm of Boolean function and , *Y* is the binary number which represent the maximum Minterm then The 1's complement of X equal Y – X.
Proof :

- 1's complement of $X = 2^n$ -1-X where n equal the number of digits of X which represent the number of variables .
- The maximum Minterm $Y= 2^n - 1$.
- From 1 and 2 , the 1,s complement of X is Y- X.

2.1 The Method for generating The Prime Implicants Terms Procedure :
1. Minterms which are included in the function are put in order from the smaller to bigger in a column way.
2. Every Minterm included in the function is subtracted from the maximum Minterm whether it (maximum Minterm) is included in the function or not (For instance the maximum Minterm of a function include 3 variables equals 7 which resulted from 2^3-1).
3. The result of step 2 should be segmented into its initial values of digits but in a decimal form as shown in table 4.

Table 5. Initial values of digits in a decimal form

The maximum minterm(MAX) for 3 variable is 2^3-1=7			
Minterms (Base)	(MAX-Minterm)	Initial Value	SET
0	7	(1) (2) (4)	0'set
1	6	(2) (4)	1'set
2	5	(1) (4)	2'set
3	4	(4)	3'set
4	3	(1) (2)	4'set
5	2	(2)	5'set
6	1	(1)	6'set
7	0	-	-

- The minterm 3 is subtracted from the maximum minterm 7 to result in 4 that could not be divided.
- The minterm 4 is subtracted from 7 to result 3 that could be divided into 1 and 2.

1. Each minterm X can be combined with each minterm Y when Y equal X plus the numbers resulted from the pervious division .
   - Minterms 3 combined with minterm 7 which result from 3+4 to form a new term 3,7(4) that becomes one variable less than the two combined minterms .
   - The number between brackets is called "reference" that determine the position of the omitted variables.
   - at the same way; minterm 4 combined with minterm 5 which result from 4+1 to from the term 4,5(1). and minterm 4 also combined with minterm 6 to form the term 4,6(2) .
2. The probabilities of minimization of the minterms included in the function are taken. This will lead to the probabilities of the double minimization.

Example:

Determine the probabilities of the double minimization of the following function:

$F(X,Y,Z) = \sum(1,2,3,4,5)$

Solution:

Table 6. The maximum minterm for 3 variable is 23-1=7

Minterms (Base)	(MAX-Minterm)	Initial Value	SET
1	6	(2)1,3 (4)1,5	1'set
2	5	(1)2,3 (4)-	2'set
3	4	(4)-	3'set
4	3	(1)4,5 (2)-	4'set
5	2	(2)-	5'set

Notice that the name of set is the name of base.

Table 7. The probabilities of double minimization

TERM	X	Y	Z	Name
(2)1,3	0	-	1	X'Z
(4)1,5	-	0	1	Y'Z
(1)2,3	0	1	-	X'Y
(1)4,5	1	0	-	XY'

♦ Quadruple Minimization

If (a)k,L is a double minimization term whose its base set is k and the follower set is L, to get the quadruple minimization look at the two sets K, L if there are one term in each set equals in reference then its Minterm can be combined with the two minterm K, L to give quadruple minimization.

Example: (1)0,1 the base set is "0" set and the follower set is "1"set are observed and compared . If there are terms equal in the reference included minterms will be taken.

This means that term (1)1,0 is combined with two minterms 2&3 because of the equality of the reference (2) in two terms (2)0,2 &(2)1,3 to result the quadruple term (1,2)0,1,2,3 at the same way , the term (1)1,0 is combined with two minterms 4 and 5 because of the equality of the reference (4) in two minterms (4)1,4 & (4)1,5 to result a quadruple term (1,4)0,1,4,5.

Table 8.

"0" set	(1)0,1 (2)0,2 (4)0,4
"1"set	(2)1,3 (4)1,5
"2"set	(1)2,3 (4)2,6

But for the quadruple minimization for the term (2) 0,2 in set "0", The term (1)0,1 neglected because it is before the term (2)0,2 in the base set "0". the term (2) 0,2 is combined with the two minterms 6&4 because of the equality of the reference (4) in the two minterms (4)2,6 &(4)0,4.

Table 9.

"0" set	(1)0,1 (2)0,2 (4)0,4
"2"set	(1)2,3 (4)2,6

Rules

1. The base which contain one term is neglected during the higher minimization.
2. All the minimization higher than the quadruple minimization for instance octal minimization is applied as the quadruple minimization.
3. It is taken in the consideration that the minterms or terms which are a part of higher minimization are neglected in the final result (The quadruple minimization for instance is higher than double minimization).

Example:

F (X,Y,Z) = (0,1,2,3,4,5)

Solution:

Table 10. The maximum minterm for 3 variable is 2^3-1=7

Minterms (Base)	(Max-Minterm)	Double minimization	Quadruple minimization	SET
0	7	(1)0,1 (2)0,2 (4)0,4	(1,2)0,1,2,3 (1,4)0,1,4,5	0'set
1	6	(2)1,3 (4)1,5		1'set
2	5	(1)2,3 (4)-		2'set
3	4	(4)-		3'set
4	3	(1)4,5 (2)-		4'set
5	2	(2)-		5'set

Example:

F (X,Y,Z,W) = (1,2,3,5,7,10,11,15)

Solution:

Table 11. The maximum minterm for 4 variable is 2^4-1=15

Minterms (Base)	(Max-Minterm)	Double minimization	Quadruple minimization	SET
1	14	(2)1,3 (4)1,5 (8)-	(2,4)1,3,5,7	1'set
2	13	(1)2,3 (4)- (8)2,10	(1,8)2,3,10,11	2'set
3	12	(4)3,7 (8)3,11	(4,8)3,7,11,15	3'set
5	10	(2)5,7 (8)-		5'set
7	8	(8)7,15		
10	5	(1)10,11 (4)-		
11	4	(4)11,15		
15	0	-		

3. IMPLEMENTATION OF BOOLEAN FUNCTION BY USING MNNs

MNNs present a new trend in neural network architecture design. Motivated by the highly-modular biological network, artificial neural net designers aim to build architectures which are more scalable and less subjected to interference than the traditional non-modular neural nets (J, Murre, (1992)).

There are now a wide variety of MNN designs for classification. Non-modular classifiers tend to introduce high internal interference because of the strong coupling among their hidden layer weights (R. Jacobs, M. Jordan, A. Barto, (1991)). As a result of this, slow learning or over fitting can be done during the learning process. Sometime, the network could not be learned for complex tasks. Such tasks tend to introduce a wide range of overlap which, in turn, causes a wide range of deviations from efficient learning in the

different regions of input space (G. Auda, M. Kamel, H. Raafat, (November, 1995)). Usually there are regions in the class feature space which show high overlap due to the resemblance of two or more input patterns (classes). At the same time, there are other regions which show little or even no overlap, due to the uniqueness of the classes therein. High coupling among hidden nodes will then, result in over and under learning at different regions. Enlarging the network, increasing the number and quality of training samples, and techniques for avoiding local minina, will not stretch the learning capabilities of the NN classifier beyond a certain limit as long as hidden nodes are tightly coupled, and hence cross talking during learning (R. Jacobs, M. Jordan, A. Barto, (1991)).

A MNN classifier attempts to reduce the effect of these problems via a divide and conquer approach. It, generally, decomposes the large size / high complexity task into several sub-tasks, each one is handled by a simple, fast, and efficient module. Then, sub-solutions are integrated via a multi-module decision-making strategy. Hence, MNN classifiers, generally, proved to be more efficient than non-modular alternatives (El-Bakry, H. M., (2001)). However, MNNs can not offer a real alternative to non-modular networks unless the MNNs designer balances the simplicity of subtasks and the efficiency of the multi module decision-making strategy. In other words, the task decomposition algorithm should produce sub tasks as they can be, but meanwhile modules have to be able to give the multi module decision making strategy enough information to take accurate global decision (G. Auda, M. Kamel, (1997)).

In previous papers (El-Bakry, H. M., (2001), El-Bakry, H. M., (2002), El-Bakry, H. M., (October 2001) & El-Bakry, H. M., (2003)), it has been shown that this model can be applied to realize non-binary data. In this paper, it is proven that MNNs can solve some problems with a little amount of requirements than non-MNNs. In section 4, XOR function, and 16 logic functions on one bit level are simply implemented using MNN. Comparisons with conventional MNN are given. In section 5, another strategy for the design of MNNs is presented and applied to realize, and 2-bit digital multiplier.

4. REDUCING HARDWARE REQUIREMENTS BY USING MNNs

In the following subsections, we investigate the usage of MNNs in some binary problems. Here, all MNNs are feedforward type, and learned by using backpropagation algorithm. In comparison with non-MNNs, we take into account the number of neurons and weights in both models as well as the number of computations during the test phase.

4.1 A simple implementation of XOR problem

There are two topologies to realize XOR function whose truth Table is shown in Table 12 using neural nets. The first uses fully connected neural nets with three neurons, two of which are in the hidden layer, and the other is in the output layer. There is no direct connections between the input and output layer as shown in Fig.1. In this case, the neural net is trained to classify all of these four patterns at the same time.

Table 12. Truth table of XOR function.

x	y	O/P
0	0	0
0	1	1
1	0	1
1	1	0

The second approach was presented by Minsky and Papert (Rumelhart, D. E., Hinton, G. E., and Williams, R. J., (1986)) which was realized using two neurons as shown in Fig. 2. The first representing logic AND and the other logic OR. The value of +1.5 for the threshold of the hidden neuron insures that it will be turned on only when both input units are on. The value of +0.5 for the output neuron insures that it will turn on only when it receives a net positive input greater than +0.5. The weight of -2 from the hidden neuron to the output one insures that the output neuron will not come on when both input neurons are on (31). Using MNNs, we may consider the problem of classifying these four patterns as two individual problems. This can be done at two steps:

1- We deal with each bit alone.
2- Consider the second bit Y, Divide the four patterns into two groups.

The first group consists of the first two patterns which realize a buffer, while the second group which contains the other two patterns represents an inverter as shown in Table 13. The first bit (X) may be used to select the function.

Table 13. Results of dividing XOR Patterns.

X	Y	O/P	New Function
0 0	0 1	0 1	Buffer (Y)
1 1	0 1	1 0	Inverter ($\overline{Y}$)

So, we may use two neural nets, one to realize the buffer, and the other to represent the inverter. Each one of them may be implemented by using only one neuron. When realizing these two neurons, we implement the weights, and perform only one summing operation. The first input X acts as a detector to select the proper weights as shown in Fig. 3. In a special case, for XOR function, there is no need to the buffer and the neural net may be represented by using only one weight corresponding to the inverter as shown in Fig. 4. As a result of using cooperative modular neural nets, XOR function is realized by using only one neuron. A comparison between the new model and the two previous approaches is given in Table 14. It is clear that the number of computations and the hardware requirements for the new model is less than that of the other models.

Table 14. A comparison between different models used to implement XOR function.

Type of Comparison	First model (three neurons)	Second model (two neurons)	New model (one neuron)
No. of computations	O(15)	O(12)	O(3)
Hardware requirements	3 neurons, 9 weights	2 neurons, 7 weights	1 neuron, 2 weights, 2 switches, 1 inverter

4.2 Implementation of logic Function using MNNs

Realization of logic functions in one bit level (X,Y) generates 16 functions which are (AND, OR, NAND, NOR, XOR, XNOR, $\overline{X}$, $\overline{Y}$, X, Y, 0, 1, $\overline{X}$ Y, X $\overline{Y}$, $\overline{X}$ +Y, X+ $\overline{Y}$). So, in order to control the selection for each one of these functions, we must have another 4 bits at the input, thereby the total input is 6 bits as shown in Table 15.

Table 15. Truth table of Logic function (one bit level) with their control selection.

Function	C1	C2	C3	C4	X	Y	O/p
AND	0	0	0	0	0	0	0
	0	0	0	0	0	1	0
	0	0	0	0	1	0	0
	0	0	0	0	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
X+ $\overline{Y}$	1	1	1	1	0	0	1
	1	1	1	1	0	1	0
	1	1	1	1	1	0	1
	1	1	1	1	1	1	1

Non-MNNs can classify these 64 patterns using a network of three layers. The hidden layer contains 8 neurons, while the output needs only one neuron and a total number of 65 weights are required. These patterns can be divided into two groups. Each group has an input of 5 bits, while the MSB is 0 with the first group and 1 with the second. The first group requires 4 neurons and 29 weights in the hidden layer, while the second needs 3 neurons and 22 weights. As a result of this, we may implement only 4 summing operations in the hidden layer (in spite of 8 neurons in case of non-MNNs) where as the MSB is used to select which group of weights must be connected to the neurons in the hidden layer. A similar procedure is done between hidden and output layer. Fig. 5 shows the structure of the first neuron in the hidden layer. A comparison between MNN and non-MNNs used to implement logic functions is shown in Table 16.

Table 16. A comparison between MNN and non MNNs used to implement 16 logic functions.

Type of Comparison	Realization using non MNNs	Realization using MNNs
No. of computations	O(121)	O(54)
Hardware requirements	9 neurons, 65 weights	5 neurons, 51 weights, 10 switches, 1 inverter

5. IMPLEMENTATION OF MORE COMPLEX FUNCTIONS BY USING MNNS

In the previous section, to simplify the problem, we make division in input, here is an example for division in output. According to the truth table shown in Table 6, instead of treating the problem as mapping 4 bits in input to 4 bits in output, we may deal with each bit in output alone. Non MNNs can realize the 2-bits multiplier with a network of three layers and a total number of 31 weights. The hidden layer contains 3 neurons, while the output one has 4 neurons. Using MNN we may simplify the problem as:

$$W = CA \tag{1}$$

$$X = AD \otimes BC = AD(\overline{B} + \overline{C}) + BC(\overline{A} + \overline{D}) = (AD + BC)(\overline{A} + \overline{B} + \overline{C} + \overline{D}) \tag{2}$$

$$Y = BD(\overline{A} + \overline{C}) = BD(\overline{A} + \overline{B} + \overline{C} + \overline{D}) \tag{3}$$

$$Z = ABCD \tag{4}$$

Equations 1, 2, 3 can be implemented using only one neuron. The third term in Equation 3 can be implemented using the output from Bit Z with a negative (inhibitory) weight. This eliminates the need to use two neurons to represent $\overline{A}$ and $\overline{D}$. Equation 2 resembles an XOR, but we must first obtain AD and BC. AD can be implemented using only one neuron. Another neuron is used to realize BC and at the same time oring (AD, BC) as well as anding the result with ($\overline{ABCD}$) as shown in Fig. 6. A comparison between MNN and non-MNNs used to implement 2bits digital multiplier is listed in Table 18.

Table 17. Truth table of 2-bit digital multiplier.

Input Patterns				Output Patterns			
D	C	B	A	Z	Y	X	W
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Table 18. A comparison between MNN and non-MNNs used to implement 2-bits digital multiplier.

Type of Comparison	Realization using non MNNs	Realization using MNNs
No. of computations	O(55)	O(35)
Hardware requirements	7 neurons, 31 weights	5 neurons, 20 weights

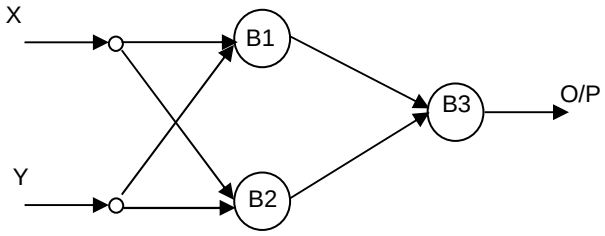

Fig. 1. Realization of XOR function using three neurons.

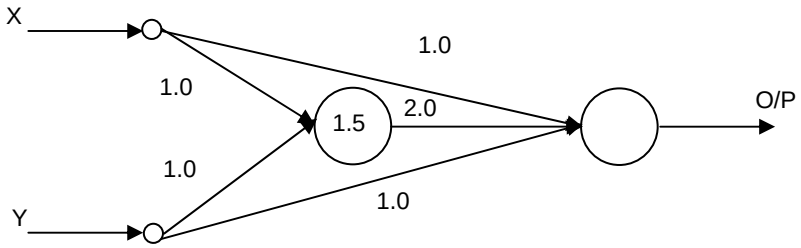

Fig. 2. Realization of XOR function using two neurons**.**

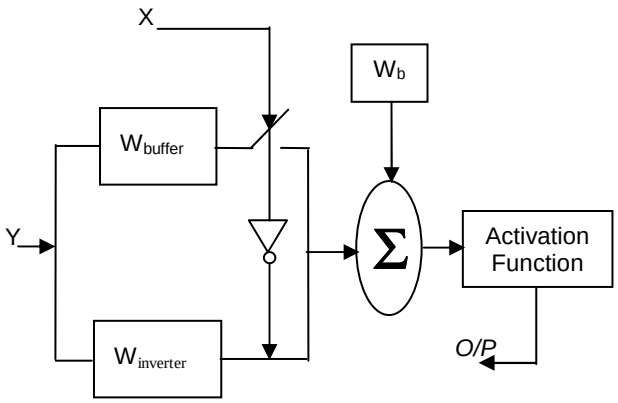

Fig. 3. Realization of XOR function using modular neural nets.

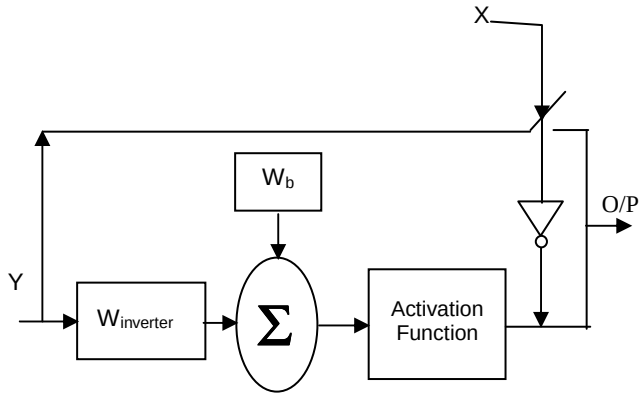

Fig. 4. Implementation of XOR function using only one neuron.

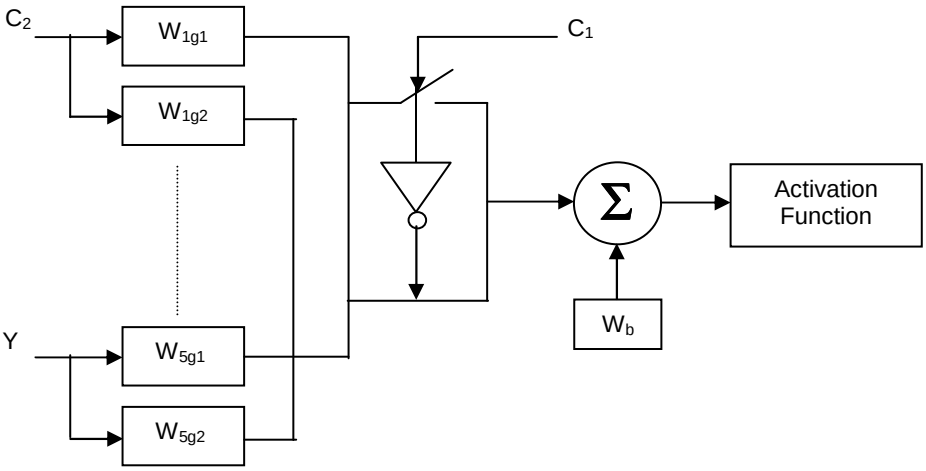

Fig. 5. Realization of logic functions using MNNs (the first neuron in the hidden layer).

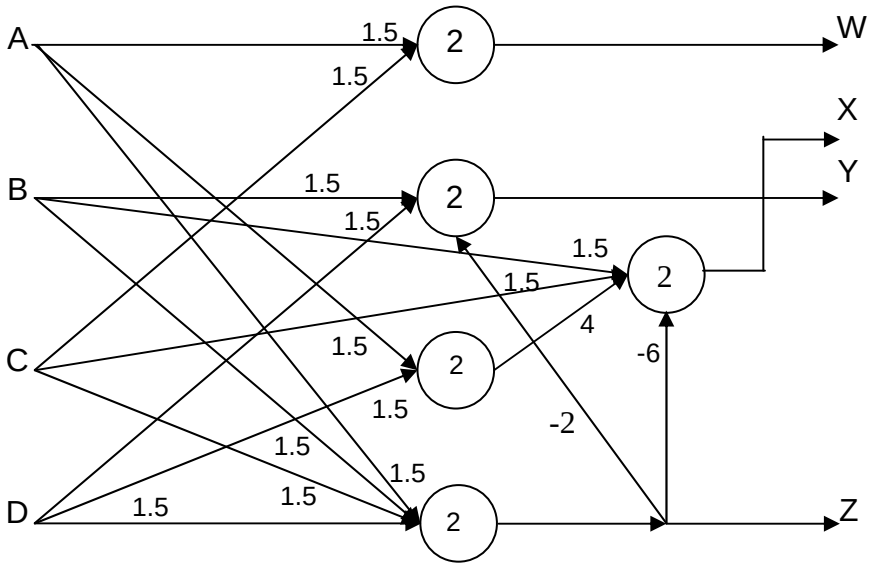

Fig. 6. Realization of 2-bits digital multiplier using MNNs.

6. CONCLUSION

A simple systematic method for generating the prime Implicants set for minimization of the Boolean functions has been introduced. Such method is a very simple because there is no need to any visual representation such as Karnough map or arrangement technique such as Tabulation method. Furthermore, it is very easy for programming. In addition, it is suitable for high variables (more than 4 variables) boolean function. Moreover, a new model for realizing complex function has been presented. Such model realies on MNNs for classifying patterns that appeared expensive to be solved by using conventional models of neural nets. This approach has been introduced to realize different types of logic functions. Also, it can be applied to manipulate non-binary data. We have shown that, compared to non MNNS, realization of problems using MNNs resulted in reduction of the number of computations, neurons and weights.

REFERENCES

Marcovitz, A. B., (2007) Introduction to Logic and Computer Design, *Hardcover*.

Marcovitz, A. B., (2005) Introduction to Logic Design, (2nd Economy Edition), *Paperback*.

Arntson, A. E., (2005) Digital Design Basics, *Paperback*.

Vahid, F., (2006) Digital Design, *Hardcover*.

Houghton, J. M., Houghton, R. S., Circuit Sense for Elementary Teachers and Students: Understanding and Building Simple Logic Circuits, *Paperback*, 1994.

Wakerly, J. F., (2005) Digital Design: Principles and Practices Package (4th Edition), *Hardcover*.

Hayes, J. P. (1993) Introduction to Digital Logic Design, *Hardcover*.

Mano, C. K., (2003) Logic and Computer Design Fundamentals (Third Edition), *Hardcover*.

Mano, M. M., Ciletti, M. D., (2003) Digital Design (4th Edition), *Hardcover*.

Mano, M. M., (1984) Digital Design, *Hardcover*.

Mano, M. M., (1992) Computer System Architecture (3rd Edition), *Hardcover*.

Biswas, N. N., (1984) Computer aided minimization procedure for boolean functions, *Proceedings of the 21st conference on Design automation*, Albuquerque, New Mexico, United States, pp. 699 – 702.

Bryant, R. E., (1986) Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, C-35-8, August, pp. 677-691.

Katz, H. R., (1986) Contemporary Logic Design/Computer Logic works Package, *Hardcover*.

Dueck, R., (2004) Digital Design with CPLD Applications and VHDL(Digital Design: Principles and Practices Package (2nd Edition), *Hardcover*.

Tomaszewski, S. P., CELIK, L. U., (1995) WWW-Based Boolean Function Minimization, *Int. J. Appl. Math. Comput. Sci*., vol. 13, no. 4, pp. 577–583.

Floyd, T. L., (2006) Electronics Fundamentals: *Circuits, Devices and Applications (7th Edition), Hardcover*.

Floyd, T. L., (1994) Digital Fundamentals, *Hardcover*.

Nelson, V. P., Nagle, H. T., Carroll B. D., Irwin, D., (1995) Digital Logic Circuit Analysis and Design, *Paperback*,

William Kleitz, (2002) Digital and Microprocessor Fundamentals: *Theory and Application (4th Edition), Hardcover*.

Crama, Y., Hammer, P. L., (January, 2006) Boolean Functions Theory, Algorithms and Applications .

http://www.asic-world.com/digital/kmaps.html, Simplification of Boolean Functions, 2006.

Atwan, A., (2006) A New Systematic Method For Simplification of Boolean Functions, *Mansoura Journal for Computer Science and Information Systems*, December, vol. 3, no. 3.

El-bakry, H. M., Mastorakis N., (2009) A Fast Computerized Method For Automatic Simplification of Boolean Functions, *Proc. of 9th WSEAS International Conference on SYSTEMS THEORY AND SCIENTIFIC COMPUTATION (ISTASC '09)*, Moscow, Russia, August 26-28, pp. 99-107.

J, Murre, (1992) Learning and Categorization in Modular Neural Networks, *Harvester Wheatcheaf*.

R. Jacobs, M. Jordan, A. Barto, (1991) Task Decomposition Through Competition in a Modular Connectionist Architecture: The what and where vision tasks, Neural Computation 3, pp. 79-87.

G. Auda, M. Kamel, H. Raafat, (November, 1995) Voting Schemes for cooperative neural network classifiers, IEEE Trans. on Neural Networks, ICNN95, vol. 3, Perth, Australia, pp. 1240-1243.

G. Auda, M. Kamel, (1997) CMNN: Cooperative Modular Neural Networks for Pattern Recognition, *Pattern Recognition Letters*, vol. 18, pp. 1391-1398.

Alpaydin, E., (1993) Multiple Networks for Function Learning, *Int. Conf. on Neural Networks*, vol. 1 CA, USA, pp. 9-14.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., (1986) Learning representation by error backpropagation, Parallel distributed Processing: Explorations *in the Microstructues of Cognition*, vol. 1, Cambridge, MA:MIT Press, pp. 318-362.

El-Bakry, H. M., (2001) Automatic Human Face Recognition Using Modular Neural Networks, *Machine Graphics & Vision Journal (MG&V)*, vol. 10, no. 1, pp. 47-73.

El-Bakry, H. M., (2002) Human Iris Detection Using Fast Cooperative Modular Neural Nets and Image Decomposition, *Machine Graphics & Vision Journal (MG&V)*, vol. 11, no. 4, pp. 498-512.

El-Bakry, H. M., (October 2001) Fast Iris Detection for Personal Verification Using Modular Neural Networks, *Lecture Notes in Computer Science, Springer, vol. 2206*, pp. 269-283.

El-Bakry, H. M., (2003) Complexity Reduction Using Modular Neural Networks, *Proc. of IEEE IJCNN'03*, Portland, Oregon, July 20-24, pp. 2202-2207.