A Mapping Technique for the Synthesis of Linear Threshold Gate Networks Used to Implement Boolean Functions

PETER CELINSKI[a], GREGORY D. SHERMAN[b], JOSÉ FCO. LÓPEZ[c] and DEREK ABBOTT[a]
[a]Department of Electrical and Electronic Engineering,[b]Department of Pure Mathematics,
Adelaide University, Adelaide, SA 5005, AUSTRALIA
[c]Research Institute for Applied Microelectronics, Universidad de Las Palmas de G.C., 35017-SPAIN
celinski@eleceng.adelaide.edu.au http://www.eleceng.adelaide.edu.au/Personal/celinski

The main result of this paper is the development of a systematic paper-and-pencil design methodology for implementing Boolean functions of up to 4 variables using threshold logic (TL) gates, which does not require linear programming, for the first time. The methodology is similar in operation to the Karnaugh map logic minimization technique, and is based on determining the minimum threshold cover of a Boolean function. The paper also reviews aspects of TL and illustrates the application of the proposed design methodology to VLSI design using the neuron-MOS technique.

Keywords: Threshold Logic, neuron-MOS, synthesis techniques, minimization, cut-complex

1 Introduction

One of the problems of designing logic circuits using threshold logic is that, unlike for conventional CMOS, there exist no general systematic techniques for the implementation of arbitrary Boolean functions in TL. More precisely, given the truth-table fully specifying a Boolean logic function, there do not exist any convenient design techniques for efficiently implementing that function using a TL gate network. The methods which until now have been used to map conventional designs to, for example, neuron-MOS, are somewhat ad-hoc and only applicable to relatively trivial functions. The aim of the paper is to develop a Karnaugh Map based mapping technique for implementing logic functions in TL, and a version of the design methodology for 3 and 4 variable circuits will be discussed.

We begin in Section 2 by giving a brief overview of threshold logic (TL). This is followed by a description of the neuron-MOS CMOS implementation. In Section 3 a review of Karnaugh-map function minimization is given, followed by a discussion on how Boolean logic functions may be computed using TL networks. Section 4 outlines the proposed method and an example of its use is given in Section 5. The translation of a TL circuit to the neuron-MOS implementation is illustrated in Section 6 and a brief conclusion follows in Section 7.

2 A Brief Overview of Threshold Logic

A threshold logic gate operates on binary variables and generates a binary output, and is functionally very similar to a hard limiting neuron, where a linear weighted sum is generated followed by a thresholding operation. The operation of a threshold gate is described by the following relations [1],

$$Y = 1 \quad \text{if} \quad \sum_{i=1}^{n} W_i X_i \geq T \tag{1}$$

$$Y = 0 \quad \text{if} \quad \sum_{i=1}^{n} W_i X_i < T. \tag{2}$$

where $X_i \in \{0, 1\}$, i = 1,. . . , n are the binary input variables, $Y \in \{0, 1\}$ is the Boolean function realized by the threshold gate, and W_i is the weight corresponding to the ith input variable X_i. T represents the gate threshold and is generally a real number satisfying

$$0 \leq T \leq \sum_{i=1}^{n} W_i. \tag{3}$$

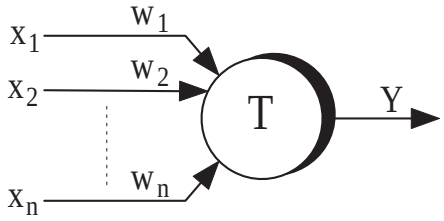

Fig. 1. A Threshold Gate model

The model of the gate is shown in Fig. 1. A Boolean function realized by such a threshold gate is called a threshold function, and a network composed of such gates is called a threshold network. Threshold functions have some useful properties. All threshold functions are Boolean functions but not all Boolean functions are threshold functions [1]. Also, all Boolean functions can be generated by a threshold gate network of depth at most two. A TL gate can be programmed to realize many distinct Boolean functions by adjusting the threshold T. For example, an n-input TL gate with W_i=1, $\forall i$ and $T = 1/2$

will realize an n input OR gate. The same gate with T set to $n - 1/2$ will realize an n-input AND gate. Generally, a threshold gate realizes a majority function, where by setting the threshold T to an appropriate value, the output of the gate Y is 1 if k or more (weighted) input variables are equal to 1, where $k \in \{0,\ldots,n\}$. The threshold gate thus offers an increased computational capability over conventional AND-OR-NOT (AON) logic, and it is therefore possible to realize Boolean functions in TL using fewer gates with a reduced logic depth [1]. Threshold logic can offer improved area density and higher speed logic circuits. Threshold logic may be loosely considered as a sub-field of neural networks. Whereas in the application of neural networks to classification the interconnection weights are modified adaptively for different inputs and the desired classification is usually only approximate, in TL the network weights are (usually) fixed and the desired function is computed exactly.

Although threshold gates are intrinsically more powerful than standard logic gates, their usefulness to integrated circuit design mainly depends on the availability of good physical realizations. The bipolar implementations of the early 70's [2] suffered from poor integration density and limited fan-in. For this reason TL has had, until recently, very little real impact in VLSI, as is evident from its gradual disappearance from textbooks on logic design. One notable early exception was the *ganged-CMOS* TL gate, formed by hard-wiring the outputs of ratioed inverters [3], but the available fan-in is limited to approximately 10 and the power dissipation is relatively high which severely limits its applicability.

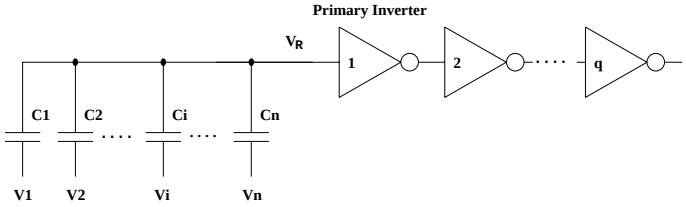

Fig. 2. The neuron-MOS gate structure

The recently developed neuron-MOS structure [4] is a CMOS TL implementation based on an array of capacitors to implement the input weights, followed by one or more inverters to implement the thresholding operation, as shown in Fig. 2. Using an odd number of inverters in the chain gives an inverting TL gate. The capacitors are usually implemented using poly1-poly2 layers. The structure in Fig. 2 is such that the `Primary Inverter` input is effectively floating, and its voltage is given by

$$V_R = \frac{\sum_{i=1}^{n} C_i V_i}{C_{tot}}, \tag{4}$$

where C_{tot} is the sum of all capacitances in the gate, including parasitic capacitances. This expression assumes that no charge is initially present on the floating node. The presence of this charge is, however, unavoidable as a result of fabrication, and for this reason techniques such as UV erasure must be used [5].

The neuron-MOS full adder cell was one of the first reported binary arithmetic circuits based on a capacitive TL gate [6] which demonstrated the potential for significant area reduction in circuits designed in neuron-MOS compared to CMOS. The neuron-MOS full adder layout reported in [6] was 55% the area of a CMOS full adder implemented in the same process, and has been shown to dissipate approximately the same amount of power as a conventional CMOS full adder [7]. The reported comparison of parallel multiplier layout areas (based on neuron-MOS and CMOS full adder cells) [6] shows that for a number of multiplicand bit lengths ranging from 16 to 64, the neuron-MOS full adder based designs have an area approximately 65% of the CMOS full adder based designs. The speed increase of neuron-MOS (7,3) parallel counter based multipliers is shown to be approximately 30% over CMOS full adder based multipliers.

Many such examples clearly illustrate the potential advantages which threshold logic offers over conventional AON logic [6], and it is with this motivation that we proceed to develop a general design methodology for implementing logic in TL.

3 Some Preliminary Observations

An n-variable or n-input Boolean function can be represented as a cube in n-dimensional Boolean space, where there is one axis for each variable, and each variable can take the values 0 or 1. To map the function onto the n-cube, we assign the value of the function corresponding to the coordinates of a vertex (ie. the value of the input variables) to each vertex of the n-cube. Fig. 3 shows the n-cube for 1 (a line), 2 (a square), 3 (a cube) and 4 (a 4-cube) dimensions.

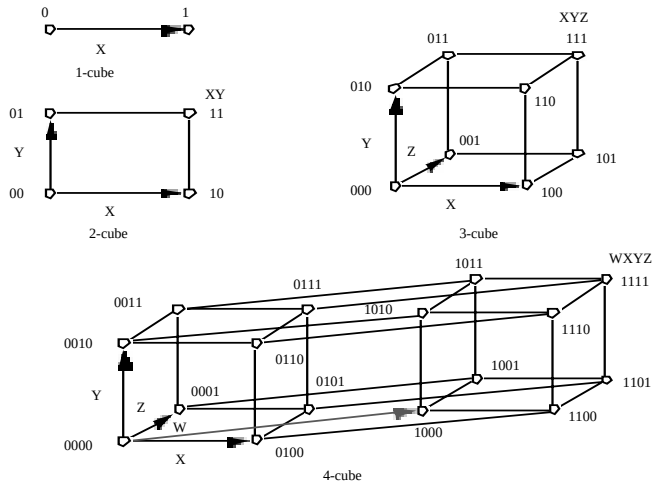

Fig. 3. N-Dimensional cubes shown for 1, 2, 3 and 4 dimensions.

The conventional Karnaugh map logic minimization method provides a means of identifying *adjacency planes* on the n-cube. Each adjacency plane corresponds to a product (AND) term of the function (in sum-of-products form), and an m-dimensional adjacency plane within an n-dimensional cube will produce a term with $n - m$ literals. In other words, the fewer planes (Kar-

naugh map groupings) the fewer the terms in the function's final (reduced) expression. Furthermore, the higher the dimension of each adjacency plane, the fewer the literals in the product term corresponding to that plane. The minimization process corresponds to finding the minimum cover of the function.

While the classical K-map minimization technique works by identifying adjacency planes and hence product terms, our technique uses the K-map to identify sets of vertices which are linearly separable from the other vertices on the n-cube and are thus "computable" by one or more threshold functions. Just as there may be more than one product term in the reduced sum-of-products expression, when using a TL function realization there may be more than one plane, and hence threshold function, required to achieve a separation of sets of vertices to compute the given function. As was mentioned earlier, a depth-2 network of threshold gates suffices to compute any Boolean function. The first layer computes the separation of each cluster of 1's from the remainder of the n-cube, and the second layer performs an OR operation on all the outputs from the first layer (using either a conventional CMOS OR gate, or a TL version). A cluster of vertices of 1's which is separated using a single hyper-plane from the rest of the vertices is called a cut-complex[8]. The number of such cut-complexes determines how many TL gates are required in the first layer of the network. An example of a function ($\bar{X}\bar{Y}\bar{Z} + YZ$) which requires two threshold gates in the first layer and the corresponding planes defined by the TL network required to compute the function is shown in Fig. 4. The black vertices correspond to logic 1's. Two techniques

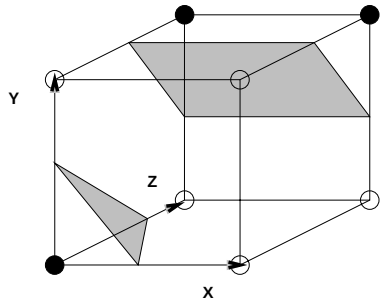

Fig. 4. An example of a function requiring two threshold gates in the first layer

which have been developed for the design of logic using TL include the "Floating Gate Potential Diagram" method [9] (which was first reported in a slightly different form by Sheng [10]), and the linear-programming approach [11]. Neither of these methods, however, is well suited to the design of TL circuits by hand, except for relatively trivial functions.

4 The Proposed Design Methodology

The technique which we have developed consists of a number of steps, and the process is somewhat similar to Karnaugh map minimization. The method will first be outlined, followed by an example which will seek to clarify the procedure.

1. The first step is to draw the Karnaugh representation of the function to be implemented in a threshold gate network. The 1's are then grouped into shapes, using the smallest number of shapes. The set of shapes is chosen such that the intersection (or union) of the shapes on the Karnaugh map covers the function precisely. The shapes are chosen from a collection of valid cut-complex shapes which are shown for up to 4-dimensions in Fig. 5. We have termed this process "finding the *minimum threshold cover* of the function".
2. If necessary, each of the cut-complex defining shapes from step 1 need to be converted into minimally-weighted cut-complexes. This is done by re-assigning the coordinates such that the shape is effectively centred about the origin (0,0,...,0) in the n-cube. This is achieved by swapping and/or inverting one or more input variables. A function is defined by codewords which represent the input values of the corresponding 1's on the Karnaugh map (ie. codeword 110 implies inputs ABC=110). The codeword weight is the sum of the number of 1's in the codeword and the weight of a shape is the sum of the codeword weights which comprise the shape. A *minimally weighted cut-complex* is one in which the weight of the shape is minimal. A minimally weighted cut-complex results in the plane for that cut-complex being such that the weights which define the plane are positive. This is necessary for implementing for example, neuron-MOS based gates.
3. The next step is to determine the planes (threshold function weights) for each minimally weighted cut-complex. This is done by choosing n orthogonal edges (where n is the number of input variables) - the midpoints of these edges then define the plane. The edges must be chosen correctly so that the convex-hull defined by the plane includes all (but no more than the number of) points in the cut-complex.
4. The weight values can then be calculated by solving the matrix equation

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \cdot\cdot \\ w_n \end{bmatrix} = T \begin{bmatrix} 1 \\ 1 \\ \cdot \\ 1 \end{bmatrix} \quad (5)$$

This may be re-written as

$$A\,\vec{w} \;=\; T\,\vec{e}. \quad (6)$$

Solving for $\vec{w}$ we obtain

$$\vec{w} \;=\; A^{-1}\,T\,\vec{e} \quad (7)$$

where A is the matrix which has as its rows the coordinates of the edge midpoints which define the plane, $\vec{w}$ is the weight vector and T is the gate threshold. If the sum of the weights is constrained to equal 1, then T is calculated as follows

$$T \;=\; (\,(\vec{e})^t\,A^{-1}\,\vec{e}\,)^{-1} \quad (8)$$

5. Finally the network may be constructed by combining the outputs of each threshold gate in the input layer into an OR gate or AND gate, depending on whether the function is defined using union or intersection of the shapes, respectively. The output of this OR or AND gate is the function output.

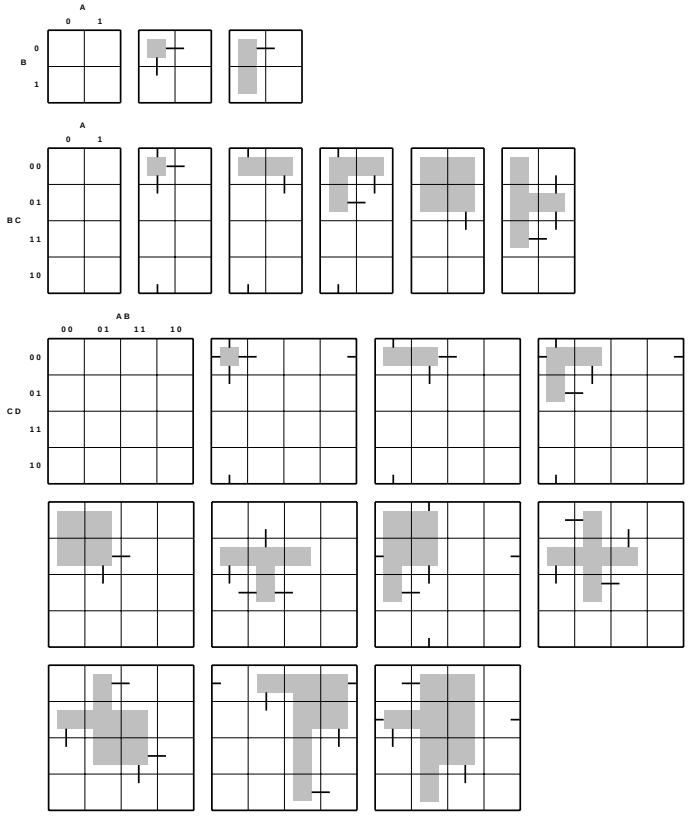

Fig. 5. All of the Karnaugh map shapes (up to isomorphism) and selected orthogonal edges for possible cut-complexes in 2, 3 and 4-dimensions.

The short black lines in Fig. 5 represent the edges which are used when choosing midpoints. The edges are chosen such that they cut the perimeter of the shape and they correspond to pairs of adjacent positions on the Karnaugh map (adjacent points on the n-cube). For those shapes which are sub-cubes of the n-cube (eg. a square in a 3-cube), one or more input variables is removed. For this reason some shapes on the 4-variable Karnaugh map are shown to have less than 4 chosen edges. The removed inputs are those which are not partitioned by a chosen edge.

Two shapes are said to be *isomorphic* if one can be derived from another by a combination of swapping and negation of inputs. A minimum weighted cut-complex is always isomorphic to its original shape. The corresponding chosen edges are also transformed by such operations. Isomorphic shapes on the Karnaugh map correspond to identical shapes on the n-cube, merely translated and/or rotated. Also, it should be noted that the complements (white shapes) of the shaded shapes in Fig. 5 are also valid shapes, with the same chosen edges as the shaded counterparts.

5 A Design Example

In this section we illustrate how the proposed method may be used to design the threshold gate network to implement Boolean functions. The example is of the 2-bit non-equivalence ($A_1A_0 \neq B_1B_0$) function. The Karnaugh map for $Y \equiv (A_1A_0 \neq B_1B_0)$ is shown in Fig. 6.

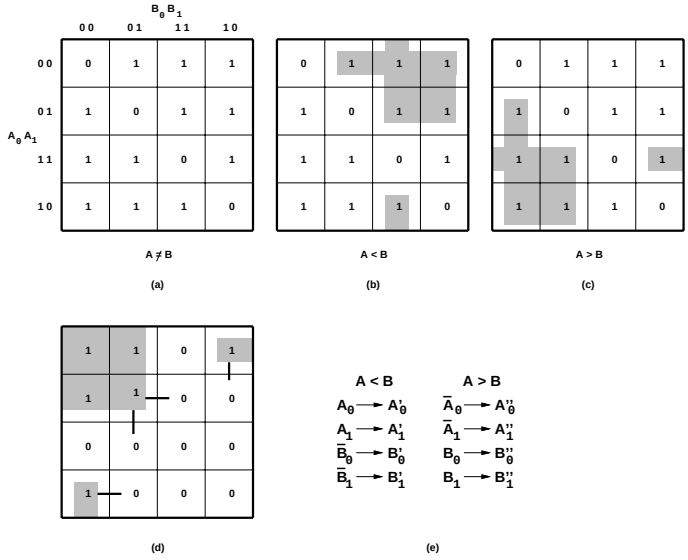

Fig. 6. (a) Karnaugh map for $Y \equiv (A_1A_0 \neq B_1B_0)$ (b) The minimum threshold cover for $A < B$ (c) The minimum threshold cover for $A > B$ (d) The minimally weighted cut-complex and selected orthogonal edges (e) The required input re-assignment for both (b) and (c) to obtain (d)

Following the procedure outlined in Section , the 1's are gouped using the minimum number of largest shapes, which in this case is two, one shape for $A < B$ and a second shape for $A > B$. The resulting minimum threshold cover is shown in Fig. 6(a) and (b). The shape used is the third to last shape shown on the 4-dimensional Karnaugh maps in Fig. 5. The cut complexes defined by the two shapes in their current positions on the map are not minimally weighted, and both shapes need to be shifted to that shown in Fig. 6(d). The resulting transformation of the inputs for both $A < B$ and $A > B$ is shown in Fig. 6(e). The four orthogonal edges are then chosen as indicated by the short thick lines in Fig. 6(d), and the coordinates of these edges are (0.5,1,0,1), (0,0.5,1,0), (0,1,0.5,1) and (1,0,0,0.5). The weight vector for both $A < B$ and $A > B$ is the same (but different input variables appear inverted in the two threshold gates as shown in the input transformation in Fig. 6(e)) and may be calculated as follows

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 0.5 & 1 & 0 & 1 \\ 0 & 0.5 & 1 & 0 \\ 0 & 1 & 0.5 & 1 \\ 1 & 0 & 0 & 0.5 \end{bmatrix}^{-1} T \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad (9)$$

As we will see later, it is sometimes desirable that the

sum of the weights equal 1, which in this case means we must set the threshold value $T = 7/16$. The weights become $w_1 = 3/8$, $w_2 = 1/8$, $w_3 = 3/8$ and $w_4 = 1/8$. The TL implementation of this gate is shown in Fig. 7. The amount of conventional AON logic which this circuit replaces is clearly quite large.

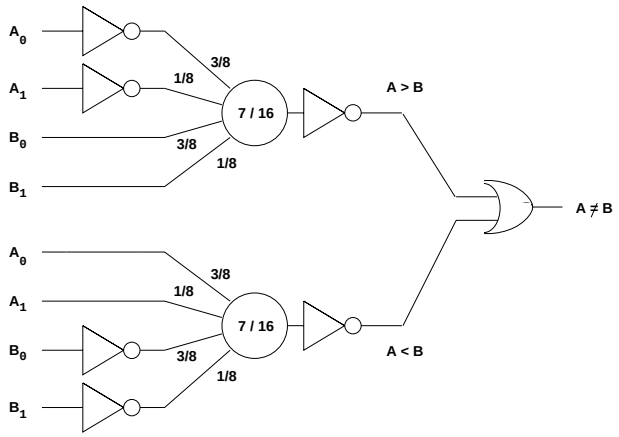

Fig. 7. The threshold logic implementation of $Y \equiv (A_1A_0 \neq B_1B_0)$

6 Mapping the TL Network to neuron-MOS

To map a TL circuit such as that shown in Fig. 7 to a neuron-MOS implementation, values for the capacitors used to implement the weights must be found, and the inverter threshold voltage of the primary inverter shown in Fig. 1(b) must be set correctly.

The ratios of the capacitances are chosen to be the same as the weights in Fig. 7 since the weights were chosen such that their sum was equal to 1 (to satisfy (4)). Using inverters sized for a threshold voltage of $7V_{DD}/16$, the neuron-MOS circuit for the network of Fig. 7 is shown in Fig. 8.

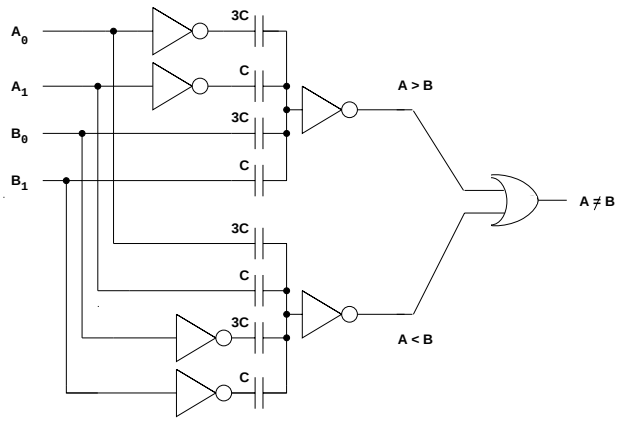

Fig. 8. The neuron-MOS implementation of $Y \equiv (A_1A_0 \neq B_1B_0)$

7 Conclusion

A methodology was developed for the systematic paper-and-pencil design of threshold gate networks implementing arbitrary Boolean functions of up to 4 variables. The operation of the design methodology was illustrated by a worked example. The mapping of a TL network designed using the proposed technique to neuron-MOS based TL gates was presented, showin the potential of the technique for developing reduced area digital systems.

Acknowledgments

The authors would like to thank Professor Mike Eastwood, from the Dept. of Pure Mathematics, Adelaide University, for many useful discussions. This work was supported by the Australian Research Council and the Sir Ross and Sir Keith Smith Fund.

References:

[1] S. Muroga, *Threshold Logic and Its Applications*, Wiley, New York, 1971.

[2] A. L. Larson, "A TTL compatible threshold gate," *IEEE JSSC*, vol. SC-8, pp. 470–471, 1973.

[3] K. J. Schultz, R. J. Francis, and K. C. Smith, "Ganged CMOS: Trading standby power for speed," *IEEE JSSC*, vol. 25, pp. 870–873, June 1990.

[4] T. Shibata and T. Ohmi, "A functional MOS transistor featuring gate-level weighted sum and threshold operations," *IEEE JSSC*, vol. 39, pp. 1444–1455, 1992.

[5] T. Shibata and T. Ohmi, "An intelligent MOS transistor featuring gate-level weighted sum and threshold operations," in *IEDM, Technical Digest*, New York, NY, USA, Dec 1991, IEEE.

[6] K. Hirose and H. Yasuura, "A comparison of parallel multipliers with neuron MOS and CMOS technologies," in *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems 96*. IEEE, November 1996, pp. 488–491.

[7] P. Celinski, D. Abbott, S.F. Al-Sarawi, and J.F. López, "Novel extension of neu-MOS techniques to neu-GaAs," *Microelectronics Journal*, vol. 31, no. 7, pp. 577–582, 2000.

[8] W. R. Emamy-K, "Geometry of cut-complexes and threshold logic," *Journal of Geometry*, vol. 65, pp. 91–100, 1999.

[9] T. Shibata and T. Ohmi, "Neuron MOS binary-logic integrated circuits - part 1 design fundamentals and soft-hardware logic circuit implementation," *IEEE Transactions on Electron Devices*, vol. 40, no. 3, pp. 570–575, March 1993.

[10] C. L. Sheng, "A graphical interpretation of realization of symmetric boolean functions with threshold elements," *IEEE Transactions on Electronic Computers*, pp. 8–18, February 1964.

[11] K. Ike, K. Hirose, and H. Yasuura, "A module generator of 2-level neuron MOS circuits," *Computers and Electrical Engineering*, vol. 24, no. 1-2, pp. 33–41, January-March 1998.