Programmable Neural Logic

Vasken Bohossian, Paul Hasler, and Jehoshua Bruck, *Senior Member, IEEE*

***Abstract*— Circuits of threshold elements (Boolean input, Boolean output neurons) have been shown to be surprisingly powerful. Useful functions such as XOR, ADD and MULTIPLY can be implemented by such circuits more efficiently than by traditional AND/OR circuits. In view of that, we have designed and built a programmable threshold element. The weights are stored on polysilicon floating gates, providing long-term retention without refresh. The weight value is increased using tunneling and decreased via hot electron injection. A weight is stored on a single transistor allowing the development of dense arrays of threshold elements. A 16-input programmable neuron was fabricated in the standard 2 μm double-poly, analog process available from MOSIS.**

We also designed and fabricated the multiple threshold element introduced in [5]. It presents the advantage of reducing the area of the layout from $O(n^2)$ to $O(n)$, (n being the number of variables) for a broad class of Boolean functions, in particular symmetric Boolean functions such as PARITY.

A long term goal of this research is to incorporate programmable single/multiple threshold elements, as building blocks in field programmable gate arrays.

***Index Terms*— Boolean functions, circuit complexity, digital logic, floating gate, hot electron injection, threshold logic, tunneling.**

I. Introduction

IN the field of neuromorphic analog VLSI, most research deals with implementing neurons that in some way learn or adapt [8], [11], [12]. That is because it is believed that the power of neural systems comes from their adaptive behavior. In fact it has been shown that the function performed by a neuron—the sum of weighted inputs followed by a threshold—is by itself (without learning) a powerful building block. For many years, theoretical computer science has studied the power of such neurons, in issues related to polynomial versus exponential size circuits and the general problem of NP completeness. The basic problem—build Boolean input Boolean output threshold circuits, to compute useful Boolean functions efficiently. Threshold circuits have been shown to be surprisingly powerful [1]. For example, integer division can be implemented by a polynomial-size threshold circuit of constant depth [3], [23]. In other words, if one is to implement a threshold circuit to compute the division of two n-bit integers, one needs polynomially many, in n, threshold elements. On the other hand, using the traditional logic circuits, composed of *AND*, *OR*, and *NOT* gates, requires exponentially many gates. That is also the case with simpler functions such as exclusive-*OR* and and integer addition.

Many results from the theory of threshold circuits could be applied to the implementation of circuits on silicon. Results such as the relationship between the maximal size allowed for the weights and the power of the resulting element or circuit [6], [9], not to mention efficient designs for *XOR, ADD, MULTIPLY*, and other useful functions, see [13], [14], and [17]. For example, a simple application of the theory led us to the introduction of a *multiple threshold element*, [5]. The latter reduces the area of the layout from $O(n^2)$ to $O(n)$ for certain Boolean functions, in particular symmetric functions, such as PARITY.

Our research has three distinct goals.

1) The implementation aspect. To design and implement efficient threshold elements on silicon.
2) The theoretical aspect. To leverage the work done in theoretical computer science in order to design high performance threshold circuits in a systematic way.
3) The programmable aspect. To introduce threshold elements as building blocks in FPGA's.

Implementations of threshold circuits were proposed already in the 60's and 70's [2], [24], [27], and more recently in [14] and [21]. To our knowledge, the theoretical results on threshold circuits have not been linked to any work involving silicon implementations. Programmable neuron-based hardware has been recently proposed [20], [22]. In the implementation section below, we show how those relate to our work. For a short overview of FPGA's see [25]. In Section II, we define the linear threshold element. In Section III, we compare threshold circuits to traditional logic circuits. In Section IV, we discuss the programmable aspect of the design. Section V shows the VLSI implementation and testing results. Finally, Section VI presents the multiple threshold element mentioned above. This element was presented in [5] from the theoretical point of view. It was compared to traditional threshold circuits and (*AND*, *OR*, *NOT*) circuits. Some of the results in [5] are summarized in Section VI which also presents an implementation of the element on a 2 μm-technology 2 mm $\times$ 2 mm chip.

II. Mathematical Setting

A linear threshold element computes a linear threshold function as shown in Fig. 1.

Manuscript received March 22, 1998; revised August 7, 1998. This work was presented in part at the Second IEEE International Conference on Innovative Systems in Silicon, Austin, TX, October 1997. This work was supported in part by the NSF Young Investigator Award CCR-9457811, by the Sloan Research Fellowship, by a grant from the IBM Almaden Research Center, San Jose, CA, by the Center for Neuromorphic Systems Engineering as a part of the National Science Foundation Engineering Research Center Program; and by the California Trade and Commerce Agency, Office of Strategic Technology.

V. Bohossian and J. Bruck are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125-0001 USA.

P. Hasler is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA.

Publisher Item Identifier S 1070-9894(98)08537-5.

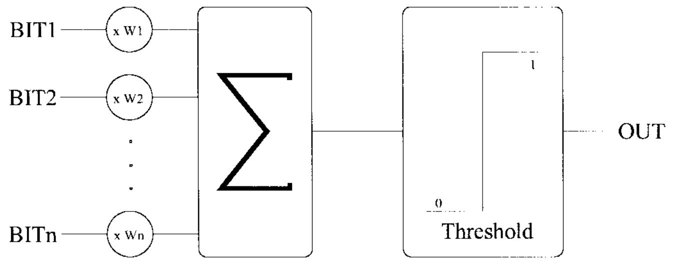

Fig. 1. Linear threshold element $y = \mathrm{sgn}(-t + \Sigma_{i=1}^{n} w_i x_i)$.

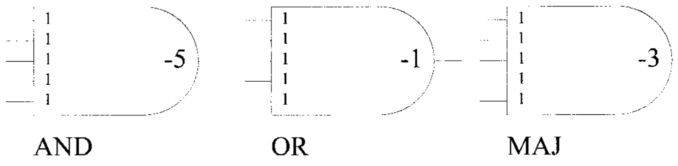

Fig. 2. Linear threshold gates for 5-input *AND*, *OR*, and Majority.

Definition 1) (Linear Threshold Function): A linear threshold function of n variables is a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ that can be written, for any input word $(x_1, \cdots, x_n) \in \{0,1\}^n$ and a fixed weight vector $(w_0, \cdots, w_n) \in Z^{n+1}$, as

$$f(X) = \mathrm{sgn}(F(X)) = \begin{cases} 1, & \text{for } F(X) \geq 0, \\ 0 & \text{otherwise} \end{cases}$$

where

$$F(X) = -w_0 + \sum_{i=1}^{n} w_i x_i.$$

Although we could allow the weights, w_i, to be real numbers, it is known [18] that for an arbitrary linear threshold function one can use integers and needs at most $O(n \log n)$ bits per weight, where n is the number of inputs.

Example 1) (AND): We want to implement *AND* of five variables. Consider the function defined by the weight vector $(w_0, \cdots, w_5) = (-5, 1, 1, 1, 1, 1)$

$$f(x_1, \cdots, x_5) = \mathrm{sgn}(-5 + x_1 + x_2 + x_3 + x_4 + x_5).$$

It outputs 1 only when all inputs are 1, therefore

$$f(x_1, \cdots, x_5) = AND(x_1, \cdots, x_5).$$

Fig. 2 shows the block representation of f along with two other Boolean functions that can be realized by a single threshold element, the conjunction, *OR*, and the majority, *MAJ*. The latter is defined in Example 2 below. □

III. Neural Logic Versus Conventional Logic

Why bother use threshold elements given that any Boolean functions can be implemented, in a systematic way, by a circuit of *AND*, *OR*, and *NOT* gates (*AON* circuit). The reason is that for some functions, such as exclusive-*OR* (*XOR*), the number of elements in the *AON* circuit will grow exponentially with the number of bits in the input, [26]. On the other hand, if one uses linear threshold elements, the number of gates is linear in the number of input bits. This is shown in Fig. 3 for a 3-bit input. In general, a depth-2, *AON* circuit computing *XOR* of n bits requires at least $2^{n-1} + 1$ gates. Using *LT*, one needs only $n + 1$ gates.

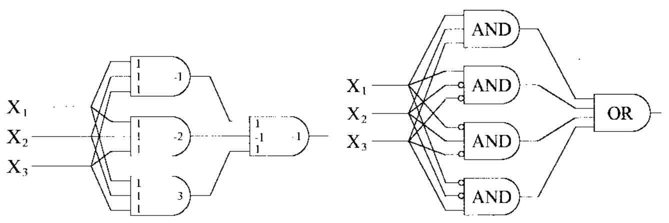

Fig. 3. Neural versus conventional logic. Two circuits computing *XOR*.

It is easy to see that *LT* circuits are more powerful than *AON* circuits. The reason is that for any single *AON* gate there is an equivalent *LT* gate, computing the same function. Example 1 shows the *LT* equivalent of *AND*. On the contrary, most *LT* gates do not have equivalents in *AON*.

Example 2: (Majority) Consider the function defined by the weight vector $(w_0, \cdots, w_5) = (-3, 1, 1, 1, 1, 1)$

$$f(x_1, \cdots, x_5) = \mathrm{sgn}(-3 + x_1 + x_2 + x_3 + x_4 + x_5).$$

It outputs 1 only when three or more of the inputs are 1. It cannot be implemented by a single *AND* or *OR* gate, even if we allow some inputs to be negated (*NOT*). □

One may argue that even though *LT* circuits are more powerful, their building blocks are more complex and therefore will require a larger area in the circuit layout. This argument is correct to some extent. However, we hope that the exponential to polynomial decrease in the number of required elements dominates the penalty introduced by an increase in their size. The following section addresses the issue.

IV. Programmable Versus Hardwired Weights

One can look at FPGA's as circuits of elements in which the function that each element computes can be programmed, that is it can be chosen among a set of available functions. In traditional FPGA's that set consists of *AND*, *OR*, and *NOT*. We propose a larger collection of functions, namely the set of Linear Threshold Functions, *LT*.

All the information about an *LT* gate is contained in the weights and threshold. We consider two ways of implementing the weights.

1) Hardwired weights are encoded in the width to length ratio of a transistor.
2) Programmable weights are stored as non volatile charge on a floating gate.

Hardwired weights cannot be changed once the circuit has been fabricated, while programmable ones can. Hardwired weights present an interesting problem in terms of automated layout. Some functions such as the comparison function, *computer*, require weights ranging from 1 to $2^{n/2}$. *AND*, *OR* and all symmetric functions can be implemented with identical weights. This difference implies that using hardwired weights, some *LT* gates are larger than others.

Using programmable weights simplifies the layout, and allows one to modify the function that the *LT* element computes. In the next section we describe the details of the implementation.

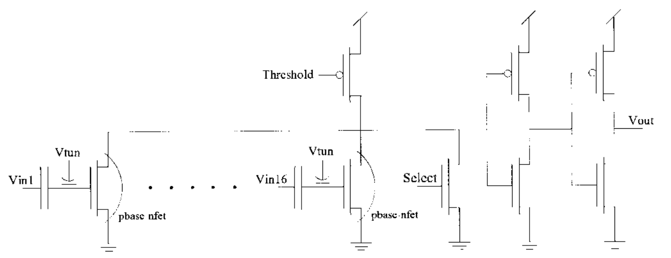

Fig. 4. Schematic of a programmable linear threshold element.

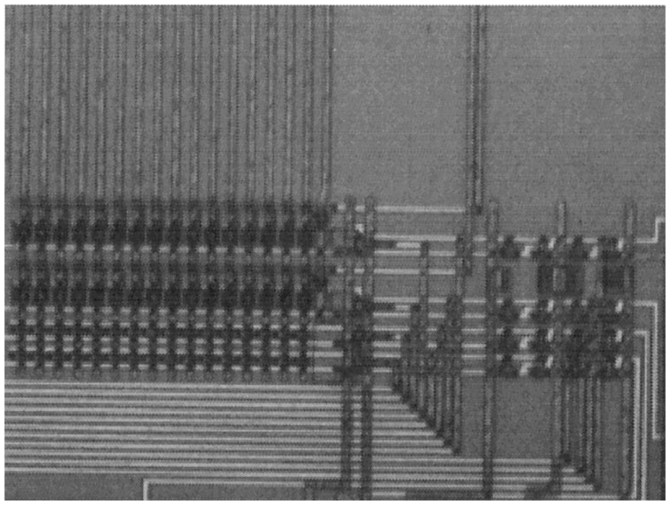

Fig. 5. Photograph of the chip area corresponding to Fig. 4. Four threshold elements are shown, two programmable (top rows) and two non programmable (bottom rows). The area shown is about 400 μm $\times$ 750 μm. The chip was fabricated using the 2 μ technology available from MOSIS.

V. Implementation and Results

In [22] the authors have fabricated a neuron-based circuit that implements an arbitrary Boolean function. We implement an arbitrary threshold element (a limited set of Boolean functions). The actual function is selected by modifying the weights. Fig. 4 shows the schematic. The threshold element consists of 16 nFET transistors with common source and drain, one pFET and two inverters. In the case of the programmable *LT* element, the 16 transistors are pbase nFET's with an isolated poly layer (floating gate). Also for the programmable case, an additional nonfloating gate nFET is included. It is used for programming the weights as explained below.

The 16-input threshold element was fabricated using the standard 2 μm double-poly, analog process available from MOSIS. Fig. 5 shows a photograph of the region corresponding to the schematic. It covers an area of about 400 μm $\times$ 750 μm. Four elements are shown. The input transistors are on the left, the inverter stage on the right. The 16 inputs (vertical lines) are fed to all four threshold elements via metal 2, such layout allows one to build dense arrays of threshold elements. Fig. 6 is a closer view on the 4 $\times$ 16 array of input transistors. Its dimensions are 200 μm $\times$ 350 μm. The top two rows correspond to the two programmable elements, while the bottom two are the fixed weight elements. Notice that the floating gate transistors (top two rows) are about twice as large as the standard ones (bottom two rows).

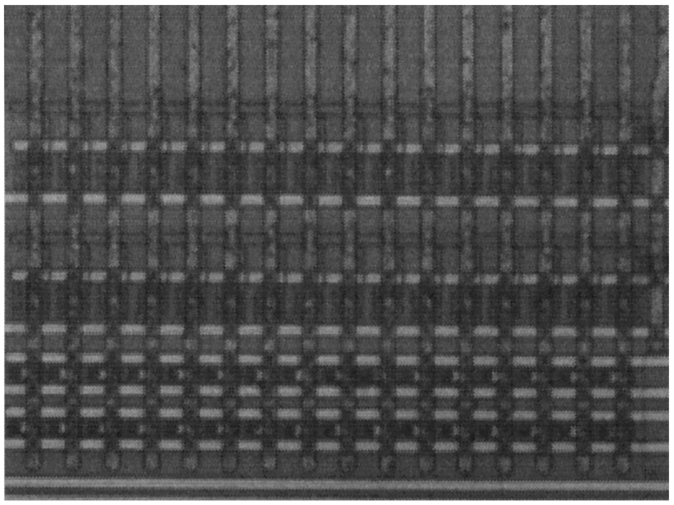

Fig. 6. Magnification of the area shown in Fig. 4 (200 μm $\times$ 350 μm); 4 $\times$ 16 transistor array. Inputs are fed through 16 vertical metal 2 lines.

A. Description of Operation

The input transistors serve as multipliers. The multiplication relies on the fact that the inputs are Boolean, 0 V for a logical 0, and X volts for a logical 1, where X can vary from 1 to 5 V. An input generates current proportional to the corresponding weight. The sum, $\Sigma_{i=1}^{n} w_i x_i$ comes naturally as we connect all transistors to the same drain and source. The threshold is subtracted using a pFET (Fig. 4). That is another difference with the approach of [21] where a capacitive sum of voltages is used, rather than a sum of currents. Finally two inverters provide hard thresholding pulling the output to logical 0, or logical 1.

B. Programming the Weights

We store the weights on polysilicon floating gates, using a single transistor per weight, providing long-term retention without refresh. To program in a new function one modifies the weights via tunneling and hot electron injection, see [11], [12], and [28] for similar applications of floating gates. There is a single tunneling line per *LT* element by means of which one can clear its weights. To program the weight separately we use hot-electron injection. For example, the weight corresponding to element i and input j (transistor (i, j) on Fig. 6), is addressed by selecting line i (Fig. 4) and input j. In other words the pins used as inputs during normal operation of the chip are also used to program in the function, no extra pins are needed (except for one select line per element).

As shown in [7] an analog memory cell, which is slightly more complex than the single transistor storage used here, can store up to 14 bits of information, an amount largely sufficient for most practical threshold functions.

C. Measurements and Discussion

Fig. 7 shows the normal operation of the *LT* element. All inputs are set to the same voltage which is varied. The pre-inverter and post inverter outputs are shown for different values of the threshold. The latter is set to 1–3, and 4 V. As expected, increasing the input voltage decreases the output.

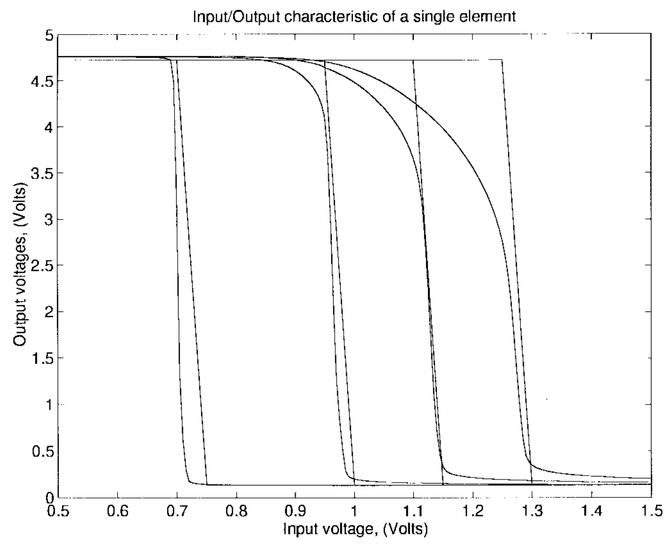

Fig. 7. Input/output operation at different values of the threshold. Threshold = 1–3, and 4 V going from right to left.

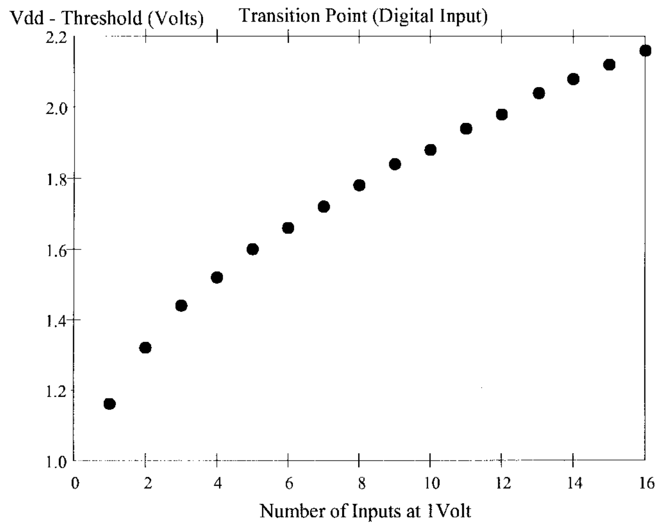

Fig. 8. $Vdd - Threshold$ versus the number of 1's in the input.

The lower the voltage on the threshold pin (corresponding to a higher threshold of the *LT* element), the higher the voltage needed on the input pins, to trigger a transition. The post-inverter response is sharper.

We tested the linearity of our threshold element by detecting the value of the threshold, w_0, at which $w_0 + \Sigma_{i=0}^{16} x_i = 0$, while varying the number of 1's in the input vector. 1 V was used as the value of logical 1. Fig. 8 shows the result. Notice the square root shape of the data. This illustrates an important point, the voltage one needs to apply in order to get a certain value of T is not linear in T. For an *nFET*, operating above or below threshold the contributions of a single input are, respectively

$$I = \frac{\beta}{2}(V_g - V_T)^2$$
$$I = I_0 e^{\kappa V_g / V_T}$$

where V_T is the thermal voltage and β, I_0, and κ are constants.

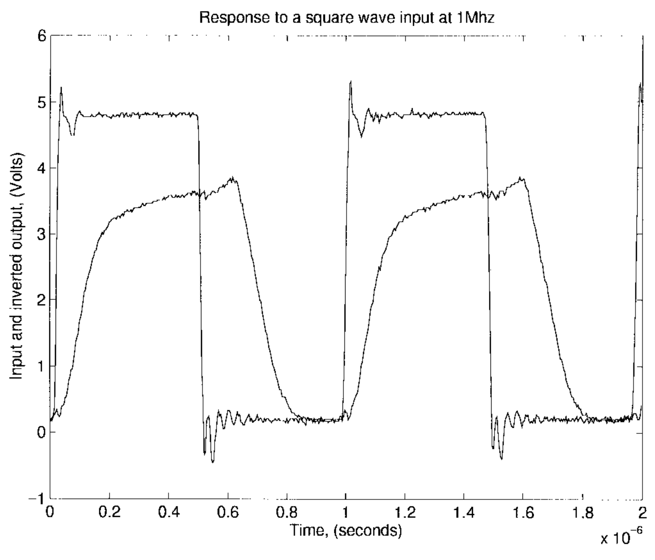

Fig. 9. Response to a 1 MHz square wave input on all 16 pins.

Hardwired weights are encoded as the W/L ratio of the transistor to which both β and I_0 are proportional [15]. That in turn makes the values of the weights linear in W/L irrespective of the region of operation of the transistor. In the case of programmable weights, the value of the weights can be quadratic or exponential in the voltage stored on the floating gate (see Fig. 8). Such nonlinearities result in a large dynamic range.

The maximum clock rate was found to be 1 MHz. Both the input and output are shown. The output is taken after the first inverter (see Fig. 4). The 16 input pins are all connected to the same input signal. The output signal is attenuated and lags behind the input. That is also due in part to the pads used. Fig. 9 shows the response.

The static power dissipation depends on the particular value of the inputs and threshold. By varying them a power dissipation of the order of 1 mW was observed. The maximal power dissipation occurs at values of X such that the sum $\Sigma\, w_i x_i$ is close to the threshold. At those values we also may get an unstable behavior; noise may bring the output to either logical 0 or 1. In general in such situations the circuit-delay is high, since it takes a long time for the output to stabilize. One can avoid this problem by selecting the weights, w_i in such a way that the above situation never occurs. That is, for all inputs $X, |\Sigma\, w_i x_i| > \epsilon$, where ϵ is the margin. For more details on how to set the margin see [4].

The above measurements are meant to provide a qualitative characterization of the prototype. In our initial implementation no steps were taken in order to optimize parameters such as power dissipation, speed and noise margin. For example, using larger inverter transistors can increase the speed of the circuit, at the expense of power.

VI. LTM: LT Element with Multiple Thresholds

In [5], the authors introduce a new computing element based on the linear threshold element. It can be viewed as a multiple threshold neuron, see [10] and [19]. Instead of the sign function in the LT element it computes an arbitrary

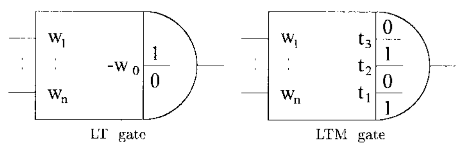

Fig. 10. Block representation of *LT* and *LTM* computing elements.

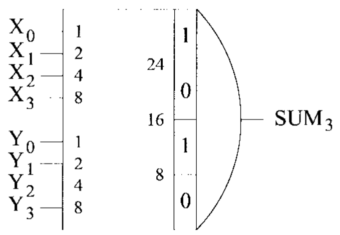

Fig. 11. Addition of two 4-bit integers, using a single layer of *LTM* gates. Only bit 3 is shown.

(with polynomialy many transitions) Boolean function of the weighted sum of its inputs (see Fig. 10).

What is the advantage of *LTM* with respect to *LT*? We show that *LTM* circuits are more amenable in implementation than *LT* circuits. In particular, the area of the VLSI layout is reduced from $O(n^2)$ in *LT* circuits to $O(n)$ in *LTM* circuits, for n input symmetric Boolean functions.

Definition 2: (LT gate with multiple transitions—*LTM*)

A function f is in *LTM* if there exists a set of weights $w_i \in Z, 1 \leq i \leq n$ and a function $h: Z \longrightarrow \{0,1\}$ such that

$$f(X) = h\left(\sum_{i=1}^{n} w_i x_i\right) \quad \text{for all } X \in \{0,1\}^n.$$

The only constraint on h is that it undergoes polynomialy many transitions.

A single *LTM* element can implement the n-input parity function. A single layer produces multiple addition. Fig. 11 shows the *LTM* element used to compute bit 3 of the addition of two 4-bit integers. For more details, examples and proofs to the above claims refer to [4] and [5].

The theoretical results about *LTM* can be applied to the VLSI implementation of Boolean functions. The idea of a gate with multiple thresholds came to us as we were looking for an efficient VLSI implementation of symmetric Boolean functions. Even though a single *LT* gate is not powerful enough to implement any symmetric function, a 2-layer *LT* circuit is. The LT_2 layout of a symmetric function requires area of $O(n^2)$, while using *LTM* one needs only area of $O(n)$. Implementing a generalized symmetric function in LT_2 requires up to n *LT* gates in the first layer. Those have the same weights w_i except for the threshold w_0. Instead of laying out n times the same linear sum $\Sigma_1^n\ w_i x_i$ we do it once and compare the result to n different thresholds. The resulting circuit corresponds to a single *LTM* gate. The LT_2 layout is redundant, it has n copies of each weight, requiring area of at least $O(n^2)$. On the other hand, *LTM* performs a single weighted sum, its area requirement is $O(n)$.

One such element was fabricated on a 2 mm $\times$ 2 mm chip, using 2 μm technology from MOSIS. It has 16 inputs. The output consists of a 4-bit bus addressing a 4-bit memory cell. The weighted sum is implemented in the Neuron MOPS fashion, as a capacitive sum of voltages, see [16], [21], as opposed to a sum of currents used in the layout of the *LT* gate; Fig. 4.

VII. Conclusion

We have fabricated and tested a 16-input programmable linear threshold element using floating gates to store the weights. Such storage requires no refresh and allows the weights to be modified via tunneling and injection. We have fabricated a second chip implementing a 16-input multi-threshold element. A single multi-threshold element can implement *XOR* and integer addition. It takes advantage of the fact that some useful Boolean functions can be implemented by a two-layer *LT* circuit in which all elements of the first layer have the same weights. That allows to reduce the area from $O(n^2)$ to $O(n)$, by implementing the weighted sum only once.

We focused on a qualitative characterization of the prototype, as a proof of concept, rather than a quantitative comparison with traditional digital logic. The theoretical results in threshold logic suggest that the number of elements used in a threshold circuit is significantly smaller than the corresponding number for digital logic circuits, for certain useful Boolean functions, as the number of inputs grows. However, for practical purposes, a thorough, qualitative comparison between the "threshold element" and the "traditional digital logic" element is required.

From the practical point of view one possible extension of this research is to devise a systematic (maybe automated) way of generating the layout of threshold circuits with hardwired weights. Another direction of research is to incorporate programmable threshold elements as building blocks in FPGA's.

Acknowledgment

The authors would like to thank the reviewers for their comments and V. Koosh for helping with the testing and analysis of the chip.

References

[1] E. Allender, "A note on the power of threshold circuits," in *Proc. 30th IEEE Symp. Foundations Comput. Sci.*, 1989.
[2] J. J. Amodei, R. O. Winder, D. Hampel, and T. R. Mayhew, "Digital circuit techniques," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 1967.
[3] P. W. Beame, S. A. Cook, and H. J. Hoover, "Log depth circuits for division and related problems," in *Proc. 25th IEEE Symp. Foundations Comput. Sci.*, 1984.
[4] V. Bohossian, "Neural logic: Theory and implementation," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, June 1998.
[5] V. Bohossian and J. Bruck, "Multiple threshold neural logic," in *Proc. Neural Inform. Process. Syst. 10*, Dec. 1997.
[6] ——, "On neural networks with minimal weights," in *Proc. Neural Inform. Process. Syst. 8*, Dec. 1995.
[7] C. Diorio, S. Mahajan, P. Hasler, B. A. Minch, and C. Mead, "A high resolution nonvolatile analog memory cell," in *Proc. Int. Conf. Circuit Syst.*, Seattle, WA, 1995, vol. 3.
[8] R. Douglas, M. Mahowald, and C. Mead, "Neuromorphic analogue VLSI," *Annu. Rev. Neurosci.*, vol. 18, 1995.
[9] M. Goldmann and M. Karpinski, "Simulating threshold circuits by majority circuits," in *Proc. 25th ACM STOC*, 1993.
[10] D. R. Haring, "Multi-threshold threshold elements," *IEEE Trans. Electron. Comput.*, vol. EC-15, Feb. 1966.

[11] P. Hasler, C. Diorio, B. A. Minch, and C. Mead, “Single transistor learning synapses,” in *Proceedings of Neural Information Processing Systems 7*. Cambridge, MA: MIT Press, Dec. 1994.
[12] M. Holler, S. Tam, H. Castro, and R. Benson, “An electrically trainable artificial neural network with 10240 ‘floating gate’ synapses,” in *Proc. Int. Joint Conf. Neural Net.*, Washington, DC, vol. 2, June 1989.
[13] W. H. Kautz, “The realization of symmetric switching functions with linear-input logical elements,” *IRE Trans. Electron. Comput.*, Mar. 1961.
[14] R. Lauwereins and J. Bruck, “Efficient implementation of a neural multiplier,” *IBM Res. Rep.*, May 30, 1991.
[15] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.
[16] B. A. Minch, C. Diorio, P. Hasler, and C. Mead, “Translinear circuits using subthreshold floating-gate MOS-transistors,” *Analog Integr. Circuit Signal Process.*, vol. 9, no. 2, Mar. 1996.
[17] R. C. Minnick, “Linear-input logic,” *IRE Trans. Electron. Comput.*, Mar. 1961.
[18] M. Muroga, *Threshold Logic and Its Applications*. New York: Wiley-Interscience, 1971.
[19] S. Olafsson and Y. S. Abu-Mostafa, “The capacity of multilevel threshold functions,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10, Mar. 1988.
[20] T. Shibata, K. Kotani, and T. Ohmi, “Real-time reconfigurable logic circuits using neuron MOS transistors,” in *Proc. Int. Solid-State Circuit Conf.*, 1993.
[21] T. Shibata and T. Ohmi, “A functional MOS transistor featuring gate-level weighted sum and threshold operations,” *IEEE Trans. Electron Devices*, vol. 39, June 1992.
[22] ——, “Neuron MOS binary-logic integrated circuits—Part I: Design fundamentals and soft-hardware-logic circuit implementation,” *IEEE Trans. Electron Devices*, vol. 40, Mar. 1993.
[23] K. Siu, J. Bruck, T. Kailath, and T. Hofmeister, “Depth efficient neural networks for division and related problems,” *IEEE Trans. Inform. Theory*, vol. 39, May 1993.
[24] T. Tich Dao, “Threshold I^2L and its applications to binary symmetric functions and multivalued logic,” *IEEE J. Solid-State Circuits*, vol. SSC-12, Oct. 1977.
[25] J. Villasenor and W. H. Mangione-Smith, “Configurable computing,” *Sci. Amer.*, June 1997.
[26] I. Wegener, “The complexity of the parity function in unbounded fan-in unbounded depth circuits,” *Theoretical Comput. Sci.*, vol. 85, 1991.
[27] B. A. Wooley and C. R. Baugh, “An Integrated m-out-of-n detection circuit using threshold logic,” *IEEE J. Solid- State Circuits*, vol. SSC-9, Oct. 1974.
[28] K. Yang and A. G. Andreou, “The multiple input floating gate MOS differential amplifier: An analog computational building block,” in *Proc. IEEE ISCAS*, London, U.K., 1994, vol. 5.

Vasken Bohossian received the B.S.E. degree in electrical engineering (with honors) from McGill University, Montreal, P.Q., Canada, in 1993, and the M.S. degree in electrical engineering and the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, in 1994 and 1998, respectively.

His research interests include parallel and distributed computing, fault-tolerant computing, error-correcting codes, computation theory, and threshold logic.

Paul Hasler received the B.S.E. and M.S. degrees in electrical engineering from Arizona State University, Tempe, in 1991 and the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, in 1997.

His research interests include using floating-gate MOS transistors to build adaptive systems in silicon, investigating the solid-state physics of floating-gate devices, and modeling high-field carrier transport in Si and SiO_2.

Jehoshua Bruck (S’86–M’89–SM’93) received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion–Israel Institute of Technology, Haifa, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1989.

He is a Professor of Computation and Neural Systems and Electrical Engineering at the California Institute of Technology, Pasadena. His research interests include parallel and distributed computing, fault-tolerant computing, error-correcting codes, computation theory, and biological systems. He has an extensive industrial experience, including, serving as Manager of the Foundations of Massively Parallel Computing Group, IBM Almaden Research Center, from 1990 to 1994, a Research Staff Member at the IBM Almaden Research Center, from 1989 to 1990, and a Researcher at the IBM Haifa Science Center, Israel, from 1982 to 1985. He published more than 130 journal and conference papers in his areas of interests and he holds 21 patents.

Dr. Bruck received the 1997 IBM Partnership Award, the 1995 Sloan Research Fellowship, the 1994 National Science Foundation Young Investigator Award, the 1992 IBM Outstanding Innovation Award for his work on “Harmonic Analysis of Neural Networks” and the 1994 IBM Outstanding Technical Achievement Award for his contributions to the design and implementation of the SP-1, the first IBM scalable parallel computer. He is a member of the Editorial Board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS.