Programmable Neural Logic

Vasken Bohossian Paul Hasler Jehoshua Bruck
California Institute of Technology
Mail Code 136-93
Pasadena, CA 91125
e-mail: {vincent, bruck}@paradise.caltech.edu
paul@pcmp.caltech.edu
URL: paradise.caltech.edu/ETR.html

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. A long term goal of this research is to incorporate programmable threshold elements, as building blocks in Field Programmable Gate Arrays.

1 Introduction

In the field of neuromorphic analog VLSI, most research deals with implementing neurons that in some way learn or adapt, [7], [9], [10]. 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], [20]. 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.

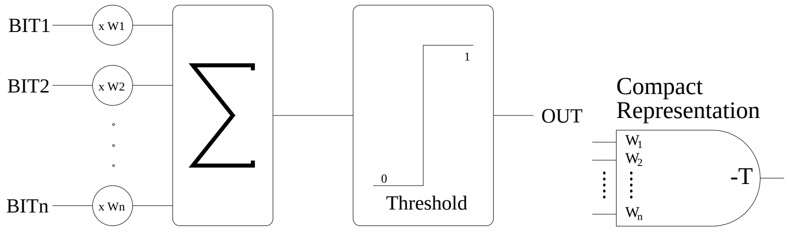

Figure 1: Linear Threshold Element $y = sgn(-t + \sum_{i=1}^{n} w_i x_i)$

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 [5], [8], not to mention efficient designs for XOR, ADD, $MULTIPLY$ and other useful functions, see [11], [13], [15].

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], [21], [24], and more recently in [13], [18]. 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 [17], [19]. In the implementation section below, we show how those relate to our work. For a short overview of FPGA's see [22]. In Section 2 we define the linear threshold element. In Section 3 we compare threshold circuits to traditional logic circuits. In Section 4 we discuss the programmable aspect of the design. Section 5 shows the implementation and results.

2 Mathematical setting

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

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

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

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

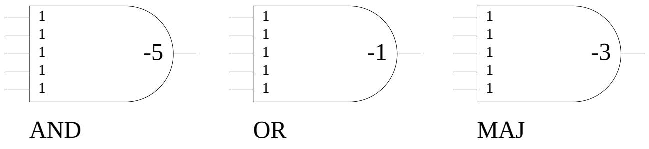

Figure 2: Linear threshold gates for 5-input AND, OR and Majority.

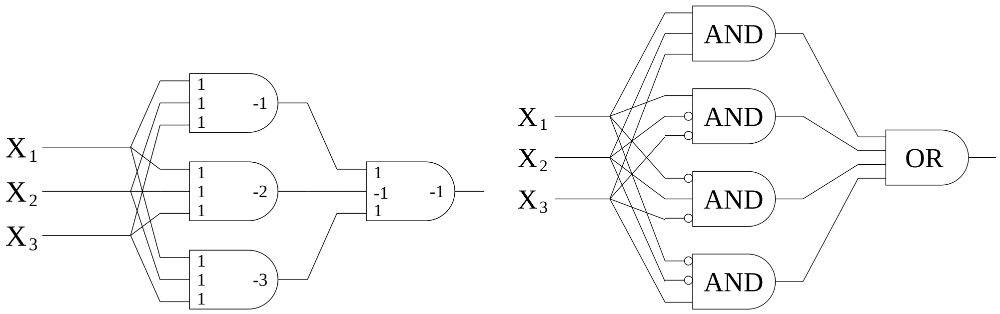

Figure 3: Neural vs. conventional logic. Two circuits computing XOR.

Although we could allow the weights, w_i, to be real numbers, it is known [16] 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 $(w_0, ..., w_5) = (-5, 1, 1, 1, 1, 1)$:

$$f(x_1, ..., x_5) = sgn(-5 + x_1 + x_2 + x_3 + x_4 + x_5)$$

It outputs 1 only when all inputs are 1, therefore:

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

Figure 2 shows the diagram for f along with two other Boolean functions that can be realized by a single threshold element. Majority is defined in Example 2 below.

3 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. 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 Figure 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.

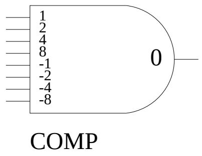

Figure 4: Comparison of two 4-bit integers.

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 $(w_0, ..., w_5) = (-3, 1, 1, 1, 1, 1)$:

$$f(x_1, ..., x_5) = sgn(-3 + x_1 + x_2 + x_3 + x_4 + x_5)$$

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, the exponential to polynomial decrease in the number of required gates dominates the penalty introduced by an increase in their size. The following section addresses the issue.

4 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.

- Hardwired weights are encoded in the width to length ratio of a transistor.
- 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, $COMP$, require weights ranging from 1 to $2^{n/2}$. Figure 4 shows a 8-bit $COMP$ function. AND, OR and all symmetric functions can be implemented with small 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.

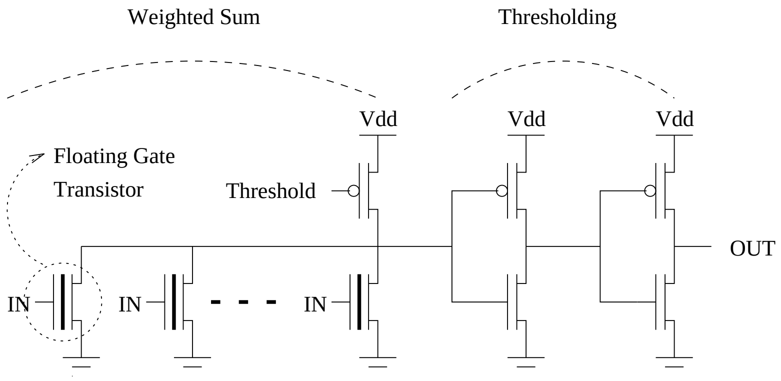

Figure 5: Schematic of a Programmable Linear Threshold Element

5 Implementation and Results

In [19] 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. Figure 5 shows the schematic implementation. A 16-input threshold element was fabricated using the standard 2 μm double - poly, analog process available from MOSIS. See Figure 6 for the layout. The 16 inputs are fed to all four gates via metal 2 (purple), such layout allows one to build dense arrays of threshold elements.

We store the weights on polysilicon floating gates, using a single transistor per weight, providing long-term retention without refresh. The multiplication relies on the fact that the inputs are boolean, 0 Volts for a logical 0, and X volts for a logical 1, where X can vary from 1 to 5 Volts. An input generates current proportional to the corresponding weight. The sum, $\sum_{i=1}^{n} w_i x_i$ comes naturally as we connect all transistors to the same node. That is another difference with the approach of [18] 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.

To program in a new function one modifies the weights via tunneling (increasing) and hot electron injection (decreasing), see [9], [10], [23] for similar applications of floating gates. As shown in [6] 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.

We tested the linearity of our threshold element by detecting the value of the threshold, w_0, at which $w_0 + \sum_{i=0}^{16} x_i = 0$, while varying the number of 1's in the input vector. 1 Volt was used as the value of logical 1. Figure 7 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:

$$I = \frac{\beta}{2}(V_g - V_T)^2$$

$$I = I_0 e^{\frac{\kappa V_g}{V_T}}$$

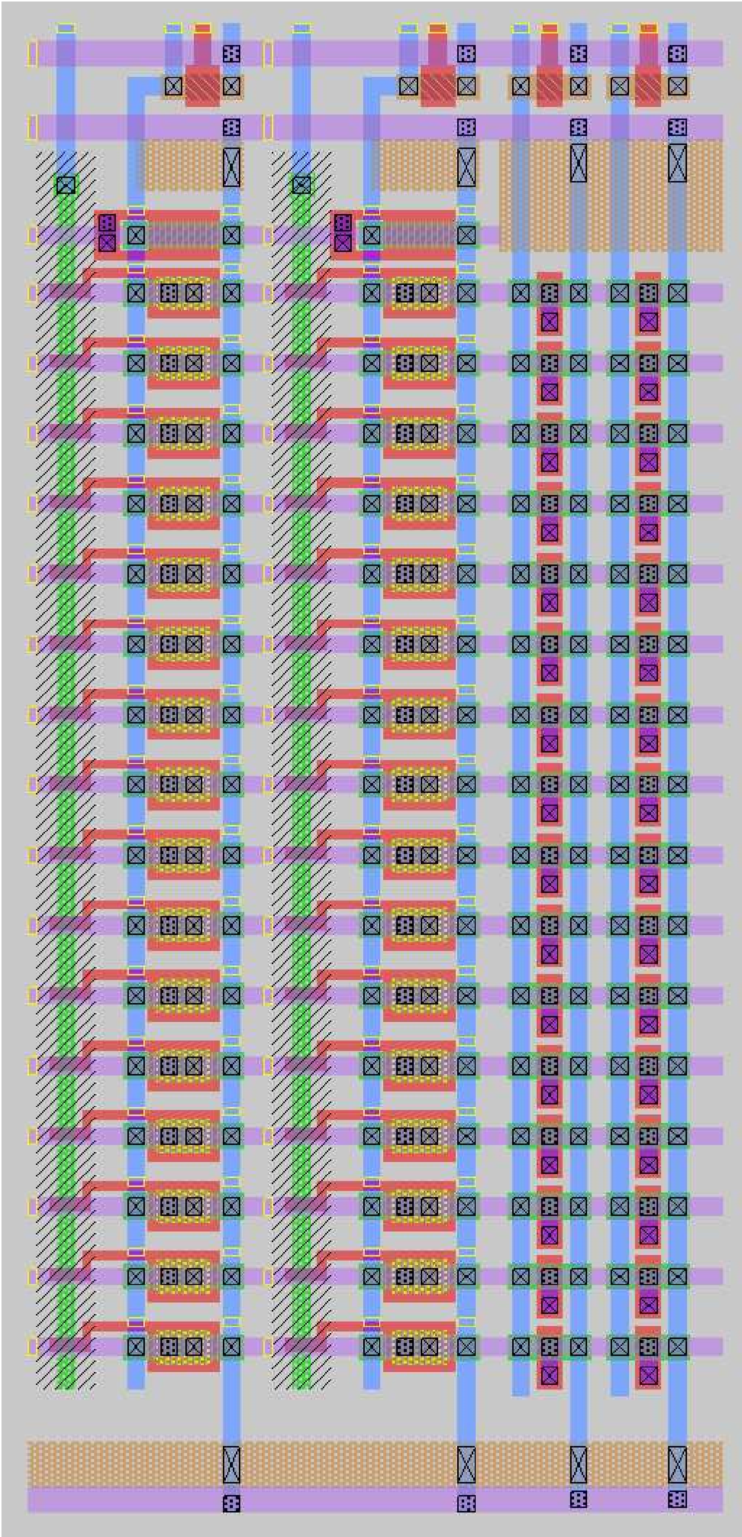

Figure 6: Layout of the linear sum $-w_0 + \sum_{i=1}^{16} w_i x_i$. Four threshold elements are shown, two programmable and two non programmable, the latter having unit weights. The Area shown is $168\mu \times 360\mu$. The chip was fabricated using the 2μ technology available from MOSIS. The dimensions of a typical contact are $4\mu \times 4\mu$.

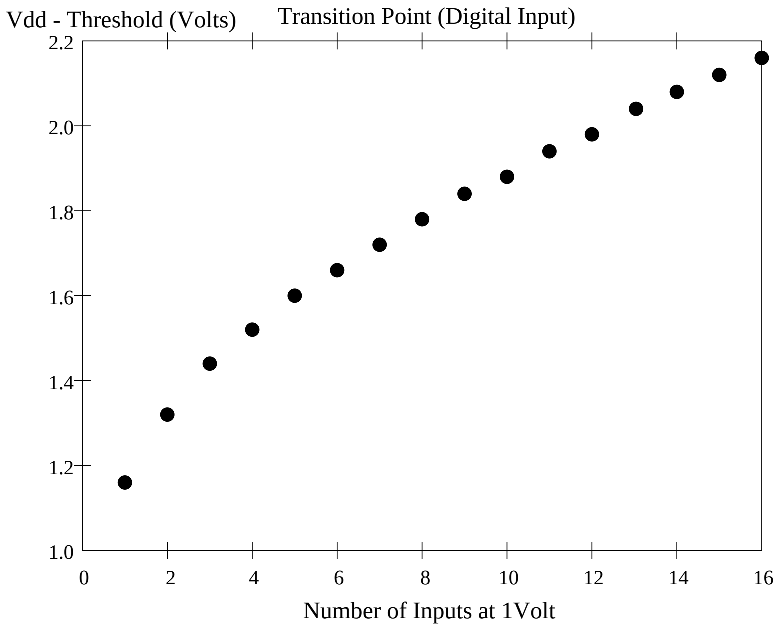

Figure 7: $Vdd - Threshold$ versus the number of 1's in the input.

where V_T is the thermal voltage and β, I_0 and κ are constants. Hardwired weights are encoded as the W/L ratio of the transistor to which both β and I_0 are proportional [14]. 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 Figure 5. Such non-linearities result in a large dynamic range.

6 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 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 2-layer LT circuit in which all gates of the first layer have the same weights. That allows to reduce the area from n^2 to n, by implementing the weighted sum only once. See [4] for further details.

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.

7 Acknowledgments

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, California, and 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. The authors would like to thank the reviewers for their comments. Special thanks to Vincent Koosh for helping with the testing and analysis of the chip.

References

[1] E. Allender. A note on the power of threshold circuits. *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 580 – 584, 1989.

[2] J.J. Amodei, R.O. Winder, D. Hampel and T.R. Mayhew. Digital Circuit Techniques *International Solid-State Circuits Conference*, February 1967.

[3] P.W. Beame, S.A. Cook and H.J. Hoover. Log depth circuits for division and related problems. *Proc. 25th IEEE Symposium on Foundations of Computer Science*, pages 1 – 6, 1984.

[4] V. Bohossian and J. Bruck. Multiple Threshold Neural Logic *Technical Report*, ETR010, June 1996. (available at http://paradise.caltech.edu/ETR.html)

[5] V. Bohossian and J. Bruck. On Neural Networks with Minimal Weights. In *Proc. of Neural Information Processing Systems 8*, 1995.

[6] C. Diorio, S. Mahajan, P. Hasler, B.A. Minch, and C. Mead, "A high resolution non-volatile analog memory cell," *Proceedings of the International Conference of Circuits and Systems*, Seattle, vol. 3, 1995, pp. 2233-2236.

[7] R. Douglas, M. Mahowald and C. Mead. Neuromorphic Analogue VLSI. *Annual Reviews in Neuroscience*, 18:255-81, 1995.

[8] M. Goldmann and M. Karpinski. Simulating threshold circuits by majority circuits. In *Proc. 25th ACM STOC*, pp. 551–560, 1993.

[9] P. Hasler, C. Diorio, B.A. Minch, and C. Mead, "Single transistor learning synapses," in *Advances in Neural Information Processing Systems 7*, MIT Press, Cambridge, MA, 1995, pp. 817-824. Also at *http://www.pcmp.caltech.edu/anaprose/paul.*

[10] M. Holler, S. Tam, H. Castro and R. Benson. An electrically trainable artificial neural network with 10240 'floating gate' synapses. *International Joint Conference on Neural Networks*, Washington, D.C., June 1989, pp. II-191 - II-196.

[11] W.H. Kautz. The Realization of Symmetric Switching Functions with Linear - Input Logical Elements. *IRE Transactions on Electronic Computers*, March 1961.

[12] M. Krause and P. Pudlak. On computing boolean functions by sparse real polynomials. *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pp. 682–691, October 1995.

[13] R. Lauwereins and J. Bruck. Efficient Implementation of a Neural Multiplier. *IBM Research Report*, RJ 8138 (74551), May 30, 1991.

[14] C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA, 1989.

[15] R.C. Minnick. Linear - Input Logic. *IRE Transactions on Electronic Computers*, March 1961.

[16] M. Muroga. *Threshold Logic and its Applications.* Wiley-Interscience, 1971.

[17] T. Shibata, K. Kotani, T. Ohmi. Real-Time Reconfigurable Logic Circuits Using Neuron MOS Transistors. *International Solid-State Circuits Conference*, 1993.

[18] T. Shibata, T. Ohmi. A Functional MOS Transistor Featuring Gate-Level Weighted Sum and Threshold Operations *IEEE Transactions on Electron Devices*, vol. 39, no. 6, June 1992.

[19] T. Shibata, T. Ohmi. Neuron MOS Binary-Logic Integrated Circuits – Part I: Design Fundamentals and Soft-Hardware-Logic Circuit Implementation. *IEEE Transactions on Electron Devices*, vol. 40, no. 3, March 1993.

[20] K. Siu, J. Bruck, T. Kailath and T. Hofmeister. Depth efficient neural networks for division and related problems. *IEEE Transactions on Information Theory*, Vol. 39(No. 3), pages 423–435, May 1993.

[21] T. Tich Dao. Threshold I^2L and Its Applications to Binary Symmetric Functions and Multivalued Logic. *IEEE Journal of Solid-State Circuits*, vol. sc-12, no. 5, October 1977.

[22] J. Villasenor and W.H. Mangione-Smith. Configurable Computing. *Scientific American*, pp. 66-71, June 1997.

[23] K. Yang and A.G. Andreou. The Multiple Input Floating Gate MOS Differential Amplifier: An Analog Computational Building Block. *IEEE ISCAS*, vol. 5, London 1994.

[24] B.A. Wooley and C.R. Baugh. An Integrated m-Out-of-n Detection Circuit Using Threshold Logic. *IEEE Journal of Solid- State Circuits*, vol. sc-9, no. 5, October 1974.