

Periodic Symmetric Functions, Serial Addition, and Multiplication with Neural Networks

Sorin Coțofană and Stamatis Vassiliadis, *Fellow, IEEE*

Abstract—This paper investigates threshold based neural networks for periodic symmetric Boolean functions and some related operations. It is shown that any n -input variable periodic symmetric Boolean function can be implemented with a feedforward linear threshold-based neural network with size of $O(\log n)$ and depth also of $O(\log n)$, both measured in terms of neurons. The maximum weight and fan-in values are in the order of $O(n)$. Under the same assumptions on weight and fan-in values, an asymptotic bound of $O(\log n)$ for both size and depth of the network is also derived for symmetric Boolean functions that can be decomposed into a constant number of periodic symmetric Boolean subfunctions. Based on this results neural networks for serial binary addition and multiplication of n -bit operands are also proposed. It is shown that the serial addition can be computed with polynomially bounded weights and a maximum fan-in in the order of $O(\log n)$ in $O(n/\log n)$ serial cycles, where a serial cycle comprises a neural gate and a latch. The implementation cost is in the order of $O(\log n)$, in terms of neural gates, and in the order of $O(\log^2 n)$, in terms of latches. Finally, it is shown that the serial multiplication can be computed in $O(n)$ serial cycles with $O(\log n)$ size neural gate network, and with $O(n \log n)$ latches. The maximum weight value in the network is in the order of $O(n^2)$ and the maximum fan-in is in the order of $O(n \log n)$.

Index Terms—Counters, feedforward neural networks, majority logic gates, McCulloch–Pitts neural networks, parity, serial binary adders, serial binary multipliers, symmetric functions, threshold logic.

I. INTRODUCTION

IN assuming threshold-based neural networks¹ the basic processing element can be a linear threshold (neural) gate² computing the Boolean function $F(X)$ such that

$$F(X) = \text{sgn}(\mathcal{F}(X)) = \begin{cases} 1 & \text{if } \mathcal{F}(X) \geq 0 \\ 0 & \text{if } \mathcal{F}(X) < 0 \end{cases}$$

$$\mathcal{F}(X) = \sum_{i=1}^n \omega_i x_i - \psi.$$

The set of input variables and of weight values associated with the inputs are defined respectively by, $X = (x_1, x_2, \dots, x_{n-1}, x_n)$ and $\Omega = (\omega_1, \omega_2, \dots, \omega_{n-1}, \omega_n)$. Such

Manuscript received February 12, 1997; revised June 9, 1998.

The authors are with the Electrical Engineering Department, Delft University of Technology, 2600 GA Delft, The Netherlands.

Publisher Item Identifier S 1045-9227(98)07011-8.

¹In this presentation for simplicity we also refer to them also as threshold networks.

²Such a threshold gate corresponds to the Boolean output neuron introduced in the McCulloch–Pitts neural model [1], [2] with no learning features. We note that currently there are some possibilities for the implementation of neural threshold devices in CMOS technology [3]–[5].

a neural gate has to contain a threshold value, ψ , a summation device, Σ , computing $\mathcal{F}(X)$ and a threshold element, T , computing $F(X) = \text{sgn}(\mathcal{F}(X))$.

It is well known that an arbitrary Boolean function can be computed using AND, OR, and NOT logical gates with no restriction in size. Given that the neural gate can also compute the logical AND, OR, and NOT [6], it can be used as the functional element for feedforward multilayer networks [6] to compute deterministically the output values of Boolean functions.

Given that Boolean functions represent the foundations for the computer-based computational paradigm, they have been the subject of numerous scientific investigations. Traditionally, Boolean functions have been implemented using Boolean networks. For a theoretical survey of Boolean networks³ implementing well known Boolean functions, the interested reader is referred to [7]. Furthermore, in the past, a number of investigations have established a number of algorithms and designs of practical importance. As a matter of fact, it is well known that a number of Boolean functions can be (and have been) implemented using a plurality of algorithms with Boolean networks; see for example, [8] for the design of frequently used arithmetic operations.

At no exception, linear threshold gate-based neural networks for the design of Boolean functions follow the paradigm of the Boolean network investigations. The investigations primary concern on minimizing one or more of four parameters, namely: the depth of the network (determined by the number of layers in the circuit), the size of the network, i.e., the number of functional elements which in the context of our discussion can be measured in terms of neural gates, the maximum number of inputs required by the functional elements, and the size of the weight values.

A special class of Boolean functions, the symmetric (and generalized symmetric) Boolean functions⁴, is encountered frequently in the realization in hardware of computer operations (e.g., error detection, arithmetic computations, etc.).

³The computation using Boolean networks, as defined in [7], requires a supply of components denoted as gates, with the gates computing some basic Boolean functions, interconnected into a system called “network” to compute one or more other Boolean functions.

⁴A Boolean symmetric function is a Boolean function for which its output value entirely depends on the sum of its input values. The Exclusive-Or (parity) of n variables is an example of a symmetric Boolean function. A generalized symmetric Boolean function is either a symmetric function or a nonsymmetric Boolean function that can be transformed into a symmetric Boolean function by trivial transformations, and can be considered to be symmetric for all theoretical purposes.

Given that symmetric (generalized or not) functions⁵ constitute a frequently used class of Boolean functions and because they are expensive to implement in hardware they have been, and continue to be, the subject of numerous theoretical and practical scientific investigations.

In this presentation we assume polynomially bounded weight and fan-in values and investigate logarithmic depth and size threshold-based neural networks for the class of symmetric Boolean functions. We prove that such networks are feasible for the class of periodic symmetric Boolean functions⁶ and for symmetric Boolean functions decomposable into a constant number of periodic symmetric subfunctions. Additionally, we investigate the benefit our results can have if used in the implementation of serial binary addition and multiplication.

This paper is organized as follows: In Section II we present related work and the main results of this investigation. In Section III we discuss some preliminary results. In Section IV we prove the main results related to periodic symmetric functions. In Sections V and VI we present new schemes for serial addition and multiplication, respectively. We conclude the presentation with some final remarks.

II. RELATED WORK AND MAIN RESULTS

In the late 1950's it has been shown by Muroga [9] that symmetric functions can be computed by depth-2 feedforward threshold networks with $(n+1)$ neural gates. This size, without increasing the depth, has been improved by Minnik [10] to be $(n/2 + 1)$ and recently in [11] it has been shown that concerning the parity function, one important symmetric Boolean function, a depth-2 network requires a size of at least $\Omega(n/\log^2 n)$ threshold gates. By increasing the depth by one, Siu *et al.* [12] have shown that Boolean symmetric functions require at most $2\sqrt{n} + O(1)$ neural gates.

The previously mentioned investigations have focused in achieving very small depth at the expense of the size of the network. One additional important question relating to the capability of producing small depth and area efficient symmetric function realizations has been investigated by Kautz [13] in the 1960's. Kautz has shown the following for the exclusive-or (parity): that the n -way Exclusive-Or can be realized by a feedforward neural network with $O(\log n)$ neural gates in $O(\log n)$ depth, with $O(n)$ weight values and number of inputs per neural gate (fan-in). Kautz's investigation however did not provide an answer to the following generic question:

- Is it possible to realize in hardware the entire class of symmetric functions by $O(\log n)$ depth and size feedforward networks with neural gates as processing elements?

As far as we know, this open question has found no answer thus far. In this paper we assume the same order of magnitude for the weight values and for the fan-in and we investigate

⁵To abbreviate the presentation, we will use in the rest of the presentation the terms symmetric functions and symmetric Boolean functions interchangeably for symmetric Boolean functions.

⁶A periodic symmetric function is a symmetric function which output values repeat after a certain interval called the function's period T , i.e., $F(x) = F(x + T)$.

this open question. We resolve it in part by addressing the realization of periodic symmetric functions, a general class of symmetric functions that includes the Exclusive-Or function. In particular we prove the following.

- Any n -input periodic symmetric Boolean function $F_p(x_1, x_2, \dots, x_n)$ with period T containing two transitions inside it and having the first positive transition at $x = \sum_{i=1}^n x_i = a$ can be implemented with an $1 + \lceil \log(\lceil (n - a)/T \rceil + 1) \rceil$ depth and size feedforward network, both measured in terms of neural gates.
- Any n -input periodic symmetric Boolean function $F_p(x_1, x_2, \dots, x_n)$ with period T containing more than two transitions inside it and having the first positive transition at $x = a_1$ can be implemented with a $t + \lceil \log(\lceil (n - a_1)/T \rceil + 1) \rceil$ depth and size feedforward network, where t is the number of neural gates necessary to implement the restriction of the function to the first period.
- Any symmetric Boolean function that can be decomposed into a constant number of periodic symmetric subfunctions can be implemented with a feedforward network with both size and depth, measured in terms of neural gates, in the order of $O(\log n)$.

Given that a number of Boolean functions present in computers⁷ belong to the class of periodic symmetric functions we further investigate the implications that our results have in the implementation of some arithmetic operations. A number of investigations have been reported regarding the implementation with threshold-based neural networks of arithmetic operations such as addition, multiplication, see, for example, [12], [14], [16], and [17], present in most hardwired computational engines. The investigations mainly concern with the upper/lower bounds for the depth of the networks (worse case delay) and the cost (the size of the network) to be expected in a realization. All known results, see, for example, [12], [14], [16], and [17], concern parallel circuit implementations.

An important class of circuit implementations, namely serial implementations of the operations, have not been addressed thus far. Serial implementations constitute an important class of circuit design in that there are numerous applications, such as signal processing [18], that require such design techniques. The reason for such requirements are usually dictated by serial data transmission, performance constraints and implementation cost. Generally speaking, serial architectures combine serial data transmission with serial computation. Given the serial reception of the operands, during the computation, intermediate results wave through the arithmetic units serially, most commonly digit by digit. Sometime the input data can also come in blocks of δ -bits at a time rather than digit by digit or bit by bit. For this type of applications and because it is more general it is more suited to refer to as δ -bit serial computations instead of bit serial or digit serial. We adopt this notation in the remaining of the presentation.

Thus far all the investigations in δ -bit serial architectures assumed logic implementation with technologies that directly

⁷They are either symmetric functions, e.g., error detection, or generalized symmetric functions [14], [15], e.g., arithmetic operations.

implement Boolean gates [19]–[23] and no studies have been dedicated to such designs using threshold-based neural networks. We assume LSB first operand reception and investigate δ -bit serial addition and multiplication in the context of feedforward neural networks. We are mainly concerned in establishing the limits of the circuit designs using threshold-based neural networks. That is we are interested in establishing theoretical bounds for delay and size of an implementation. We assume that small weight sizes are bounded at most by a polynomial size, i.e., weight values in the order of $O(n^k)$, and prove that the following holds true.

- The addition of two n -bit operands can be performed serially assuming a data transmission rate in the order of $O(\log n)$ bits per cycle, in an overall delay in the order of $O(n/\log n)$ serial cycles, with the serial cycle comprising a neural gate and a pipeline latch. The implementation cost is in the order of $O(\log n)$, in terms of neural gates, and in the order of $O(\log^2 n)$, in terms of neural latches. The weight values are polynomially bounded and the maximum fan-in is in the order of $O(\log n)$.
- The multiplication of two n -bit operands can be serially computed in $O(n)$ serial cycles. The implementation cost is in the order of $O(\log n)$, in terms of neurons, and in the order of $O(n \log n)$ in terms of neural latches. The maximum weight value is in the order of $O(n^2)$ and the maximum fan-in is in the order of $O(n \log n)$.

These results establish limits of size and delay to be expected in an important class of circuit implementations for the serial binary addition and multiplication, extensively used operations.

III. PRELIMINARY RESULTS

The class of symmetric Boolean functions S_n is a subset of B_n , the set of all n -variables Boolean functions, and it contains a number of fundamental functions, e.g., counting and sorting functions. A Boolean symmetric function is a Boolean function for which its output value entirely depends on the sum of its input values. More formally it can be defined as follows.

Definition 1: A Boolean function of n variables $F_s \in B_n$ is symmetric, i.e., $F_s \in S_n$, iff for any permutation σ of $\langle 1, 2, \dots, n \rangle$, $F_s(x_1, x_2, \dots, x_n) = F_s(x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(n)})$.

Because any input vector $X = (x_1, x_2, \dots, x_n) \in \{0, 1\}^n$ with exactly i ones is a permutation of any other vector with exactly i ones, it can be said that F_s is symmetric iff $F_s(X)$ depends only on the number of ones in the input vector X . Thus instead of the usual truth table $(X, F_s(X))$ we can describe any $F_s \in S_n$ by its spectrum $w = w_0 w_1 \dots w_n$, which is an $(n+1)$ -bit binary word with each bit w_i giving the value of F_s when exactly i input variables are one. Obviously the number of distinct spectra gives the dimension of S_n and thus there are 2^{n+1} functions in S_n .

In the following we will consider that the generic input $x = \sum_{i=1}^n x_i$ can assume one of the values $0, 1, 2, \dots, n$ in order to indicate the number of inputs $x_i, i = 1, 2, \dots, n$, which have the value one. That is to say that we can replace

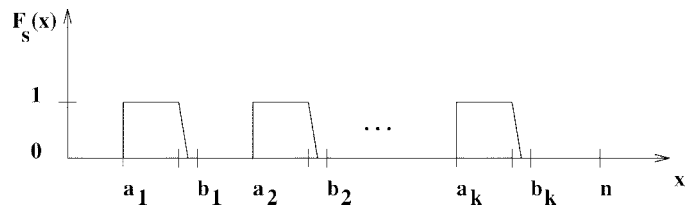


Fig. 1. Symmetric function graphic representation.

the input vector $X = (x_1, x_2, \dots, x_n)$ with an integer scalar x with its value given by $\sum_{i=1}^n x_i$ without losing any information. Consequently, any symmetric function of n variables $F_s(x_1, x_2, \dots, x_n)$ can be described by a list of those x values for which F_s is one.

Obviously x can assume only discrete values between zero and n , but as a matter of simplicity we will use the notation $x \in [k, l]$ in order to specify that x assumes values in the integer set $\{k, k+1, \dots, l\}, 0 \leq k < l \leq n$. Given that $x \in [0, n]$ and because the x values for which $F_s(x)$ is equal with one can be potentially merged in intervals we can graphically describe any symmetric function as in Fig. 1. In the figure each $a_i, i = 1, 2, \dots, k$ defines a positive transition of F_s , each $b_i, i = 1, 2, \dots, k$ defines a negative transition of F_s and $a_1 < b_1 < a_2 < b_2 < \dots < a_k < b_k$. That is equivalent with saying that $F_s(x)$ is one for all the input combinations x for which $\exists i \in [1, k]$ such that $x \in [a_i, b_i - 1]$ and zero otherwise.

An other suggestive way of representing symmetric functions can be derived by linking the graphical representation depicted in Fig. 1 with a formal expression. Thus any symmetric function can be expressed as a “sum” of the intervals in which F_s has the value one

$$F_s(x) = a_1^{\pm} b_1^{\mp} + a_2^{\pm} b_2^{\mp} + \dots + a_k^{\pm} b_k^{\mp} \quad (1)$$

where $a_i^{\pm} = 1$ if $x \geq a_i$, for $i = 1, 2, \dots, k$, $b_i^{\mp} = 1$ if $x < b_i$, for $i = 1, 2, \dots, k$ concatenation means logic AND and + means logic OR.

Given that we have introduced the basic definitions and concepts we can initiate our investigations concerning the implementation of symmetric functions with feedforward neural gate-based networks by proving some preliminary results. In the remaining of this section we focus our investigation and proofs on feedforward neural networks which follow the structure of the networks proposed by Kautz [13] or by Minnick [10],⁸ i.e., in which any gate receives as inputs at least the input vector X .

Property 1: Any symmetric function $F_s(X)$ of n variables, $X = (x_1, x_2, \dots, x_{n-1}, x_n)$, can be described with an expression like (1) with at most $\lceil (n+1)/2 \rceil$ terms (intervals).

Proof: Any symmetric function of n variables is characterized by a spectrum w with $n+1$ elements. That is to say it can be graphically represented in the input domain $[0, n]$. In order to define an interval in the graphical representation of $F_s(x)$ we need two different numbers a_i and b_i as it can be observed in Fig. 1. Each number between zero and n can

⁸We note here that with some additional precautions our results can be extended and proved also for other feedforward neural-network structures.

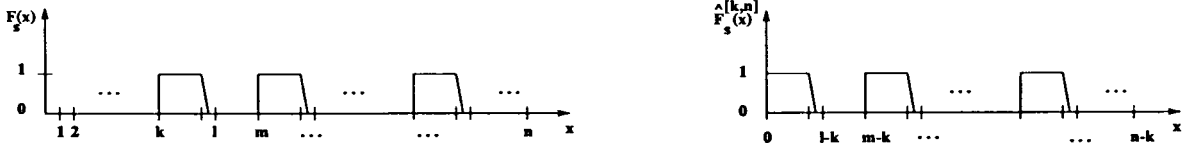


Fig. 2. Restriction of $F_s(x)$ to $\hat{F}_s^{[k,n]}(x)$.

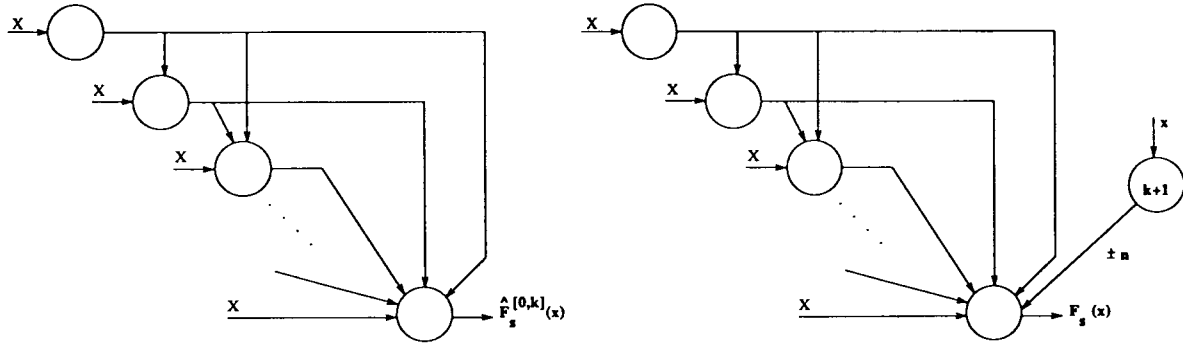


Fig. 3. Modification of the $\hat{F}_s^{[0,k]}(x)$ network in order to implement $F_s(x)$.

be used at most in the definition of only one interval, either as a_i or as b_i , because otherwise the intervals overlap or join. Therefore, it is not possible to define more than $\lceil (n + 1)/2 \rceil$ distinct intervals. ■

Definition 2: Given a symmetric function $F_s(x)$ of n variables specified by a spectrum $w = w_0w_1 \cdots w_n$ it is always possible to define a restriction of $F_s(x)$ to the input interval $[k, l]$, $\hat{F}_s^{[k,l]}(x)$, for any $k, l, 0 \leq k < l \leq n$. $\hat{F}_s^{[k,l]}(x)$ is a symmetric function of $(l - k)$ variables that has an $(l - k + 1)$ -bit spectrum $\hat{w} = \hat{w}_0\hat{w}_1 \cdots \hat{w}_{l-k}$ deduced from the initial spectrum w as $\hat{w}_i = w_{i+k}$ for $i = 0, 1, \dots, l - k$.

Lemma 1: Any symmetric function $F_s(x)$ of n variables and with the first positive transition at $x = k$ can be implemented with neural gates at the same cost as the restriction of $F_s(x)$ to the input interval $[k, n]$, $\hat{F}_s^{[k,n]}(x)$, which is a symmetric function of $(n - k)$ variables.

Proof: Any symmetric function $F_s(x)$ of n variables can be characterized by its spectrum $w = w_0w_1 \cdots w_n$. The fact that the first positive transition is positioned at $x = k$ means that the spectrum bits $w_kw_{k+1} \cdots w_n$ give all the information about $F_s(x)$. Consequently, it is enough to memorize these bits and the value of k in order to have a complete description of $F_s(x)$. As a matter of fact the spectrum restriction $\hat{w} = w_kw_{k+1} \cdots w_n$ is an $(n - k + 1)$ -bit word, i.e., the spectrum of an other symmetric function $\hat{F}_s^{[k,n]}(x)$ of $(n - k)$ variables. As it can be observed in Fig. 2, the restriction of $F_s(x)$ to $\hat{F}_s^{[k,n]}(x)$ is equivalent with a translation of the origin from zero to k . Thus a feedforward neural network that implements $F_s(x)$ can be constructed from the feedforward neural network that implements $\hat{F}_s^{[k,n]}(x)$ by adding k to each threshold value assigned to the neural gates that compose the network in order to translate the intervals back in their correct positions. There is no need to modify also the weight values associated to the gate inputs because the spectrum restriction process preserves the distances between intervals and the lengths of the intervals.

Thus the implementation cost which applies for $\hat{F}_s^{[k,n]}(x)$ is maintained also for $F_s(x)$ because we do not have to add any extra neural gate to the network implementing the restriction.

Lemma 2: Any symmetric function $F_s(x)$ of n variables and with the last transition at $x = k$ can be implemented with neural gates at the implementation cost of the restriction of $F_s(x)$ to the input interval $[0, k]$, $\hat{F}_s^{[0,k]}(x)$, which is a symmetric function of k variables, plus at most an extra neural gate.

Proof: The positioning of the last transition at $x = k$ means that the spectrum bits $w_{k+1}, w_{k+2}, \dots, w_n$ do not provide any additional information about $F_s(x)$. Thus it is enough to memorize only the first $(k + 1)$ bits of the spectrum w in order to have a complete description of $F_s(x)$. The spectrum restriction $\hat{w} = w_0w_1 \cdots w_k$ is a $(k + 1)$ -bit word, i.e., the spectrum of an other symmetric function $\hat{F}_s^{[0,k]}(x)$ of k variables. If we construct a feedforward network which implements $\hat{F}_s^{[0,k]}(x)$ it obviously implements also $F_s(x)$ for $x \in [0, k]$ but it might produce some spurious transitions for $x > k$. In order to cut out these unwanted transitions it is enough to connect an extra neural gate with the threshold $(k + 1)$ to the last neural gate of the feedforward network that implements $\hat{F}_s^{[0,k]}(x)$. This modification is graphically depicted⁹ in Fig. 3. The weight value for this new input added to the last neural gate in the network has to be n , if the last transition in the specification of $F_s(x)$ is positive, or $-n$, if the last transition in the specification of $F_s(x)$ is negative. This new element will force the global output of the network to one if the associated weight is n or to zero if the associated weight is $-n$ for any $x, k < x \leq n$. ■

Lemma 3: The implementation cost, measured in terms of neural gates, of any n variables symmetric function $F_s(x)$

⁹We note here that the weight and the threshold values assigned to the gates composing the feedforward network that implements $\hat{F}_s^{[0,k]}(x)$ were not specified in the figure because they are not relevant in this context.

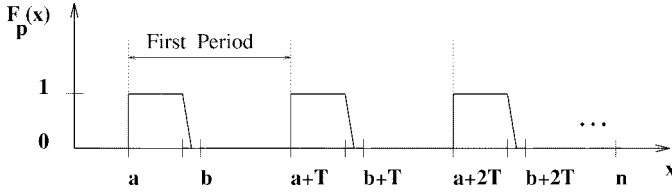


Fig. 4. Periodic symmetric function.

defined with k intervals, with the first transition at $x = m$, and the last transition at $x = M$ is the same as the implementation cost of the restriction of $F_s(x)$ to the input interval $[m, M]$, $\hat{F}_s^{[m, M]}(x)$, i.e., a symmetric function of $(M-m)$ input variables and with the same number of intervals k , plus at most one neural gate.

Proof: The proof is immediate as a result of successive application of Lemmas 1 and 2. ■

Thus far we have investigated cost related properties that apply to the implementation with feedforward neural networks of the entire class of symmetric functions. In the section to follow we restrict our investigation to periodic symmetric function. In particular we assume feedforward neural gate based networks with weight and fan-in values in the order of $O(n)$ and investigate the possibilities they can offer in the implementation of periodic symmetric function.

IV. PERIODIC SYMMETRIC FUNCTIONS

Consider the symmetric function $F_s(x)$ graphically represented in Fig. 1 and assume that the transitions are settled such that

$$\begin{aligned} a_i &= a + (i-1) \times T \\ b_i &= b + (i-1) \times T \end{aligned}$$

for all $i, i = 1, 2, \dots, i_{\max}$ and a given integer constant T . This assumption on the transitions means actually that the function repeats its value if the generic input x increases with a multiple of T , i.e., the function has a periodical behavior. More formally we can define a periodic symmetric function $F_p(x)$ as a symmetric function for which $\exists T$, an integer constant called the function's period, such that $F_p(x + kT) = F_p(x)$, for any $x + kT \in [0, n]$, $x \in [0, T-1]$, and $k = 0, 1, \dots, \lceil n/T \rceil$. Such a symmetric function is graphically depicted in Fig. 4.

Obviously for any periodical symmetric function $F_p(x)$, defined as in the previous paragraph, it is enough to know T , the value of the period, the value a of the input variable x corresponding to the first positive transition, and the x value for the first negative transition b in order to achieve a complete description of $F_p(x)$. Generally speaking, in order to implement such a function, we have to achieve two goals: first we have to construct a network that produces the correct output for the first function period; consequently, we have to extend this network such that it covers all the function transitions within the function definition domain $[0, n]$. In the following theorem we will use this approach and introduce a logarithmic depth and size feedforward neural network for periodic symmetric functions.

Theorem 1: Any periodic symmetric function $F_p(x_1, x_2, \dots, x_{n-1}, x_n)$ with the period T and two transitions inside the first period specified by a and b can be implemented with a feedforward network having $r = 1 + \lceil \log(\lceil n - a/T \rceil + 1) \rceil$ size and depth, and the values of the thresholds and the weights associated with the neural gates given by

$$\begin{aligned} \beta_{r,r} &= a, \quad \beta_{r-1,r-1} = b \\ \beta_{i,i} &= \beta_{i+1,i+1} + T2^{(r-(i+2))} \\ & \quad i = r-2, r-3, \dots, 2, 1 \\ \beta_{i,k} &= -T2^{(r-(i+1))}, \quad i < r, k = i+1, i+2, \dots, r-1, r. \end{aligned}$$

Proof: The network we propose has a similar structure to the network used by Kautz [13] in the implementation of the n -way Exclusive-Or and it is depicted in Fig. 5 where the generic input χ designates the entire set of inputs $\{x_1, x_2, \dots, x_n\}$, each one with weight one. Now we have to prove that given the values we assumed for the β s, the output of the network follows the value assumed by $F_p(x)$ for all the possible values of the input variable x .

As a consequence of Lemma 1, F_p can be implemented at the same cost as its restriction $F_p^{[a,n]}$. Because each period contains one interval $F_p^{[a,n]}$ can be specified with an expression like (1) that contains $\lceil (n-a)/T \rceil$ intervals. This means that also F_p has only $\lceil (n-a)/T \rceil$ positive transitions. As was suggested in [13], for this type of feedforward networks the number of positive transitions at the output of the element r is at most 2^{r-1} . This upper bound is imposed by the fact that the previous $r-1$ element outputs can be combined in order to yield different apparent thresholds for the element r in at most 2^{r-1} ways. Because after the last positive transition which corresponds to $x = \sum_{i=1}^r \beta_{i,r}$ no negative transition can occur the number of intervals that can be covered by a network with r elements is $2^{r-1} - 1$. Consequently $2^{r-1} - 1 = \lceil (n-a)/T \rceil$ and this equation leads to $r = 1 + \lceil \log(\lceil (n-a)/T \rceil + 1) \rceil$ neural gates and no feedforward network structured like the network in Fig. 5 with fewer number of elements (levels) can implement F_p .

We still have to prove that the algorithm we proposed for the computation of the weight and threshold values settles the 2^{r-1} positive transitions in the right positions, i.e., at $x = a + kT$, and also the negative transitions at $x = b + kT$, for $k = 0, 1, \dots, \lceil (n-a)/T \rceil - 1$. In the general case, from the 2^{r-1} possible positive transitions at the output of the r th element, only r transitions are directly settled by the values of $\beta_{i,r}, i = 1, 2, \dots, r$. The remainder of $2^{r-1} - r$ positive transitions appear at the x values that are linear combinations of the x values that correspond to these independent positive transitions. In our case, as it can be observed in Fig. 5, because of the fact that $\beta_{r,r} = a$ and $\beta_{i,r} = -T2^{(r-(i+1))}, i = 1, 2, \dots, r-1$, the r arbitrary transitions are settled at $a, a + 2^0T, a + 2^1T, a + 2^2T, \dots, a + 2^{r-2}T$. Consequently, the dependent transitions appear at all linear combinations of the independent ones and it follows that we have transitions also at $a + (2^0 + 2^1)T, a + (2^0 + 2^2)T, \dots, a + (2^0 + 2^1 + 2^2 + \dots + 2^{r-2})T$ and thus we have positive transitions at all $x = a + kT, k = 0, 1, \dots, \lceil (n-a)/T \rceil - 1$. The fact that

$O(n)$. Concerning the maximum weight value obviously it can not be greater than $T2^{\log n}$, therefore it is also in the order of $O(n)$.

Another way of expressing the generalized periodic symmetric function depicted in Fig. 6 is as an OR sum of k periodic symmetric subfunctions $F_p^i(x), i = 1, 2, \dots, k$, each one having the period T and two transitions inside the first period given by a_i and b_i . We can further relax the constraint that all the subfunctions in which F_p is decomposed should have the same period. Consequently, the function we consider is no longer periodic but it can be expressed as $\cup_{i=1}^l F_p^i(x)$, where each $F_p^i(x)$ is a periodic function with two transitions per period. In the following theorem we will extend the class of symmetric functions that can be implemented in logarithmic depth and cost from the class of periodic symmetric functions to the class of symmetric functions decomposable into a constant number of periodical harmonics.

Theorem 3: Any symmetric function $F_s(x_1, x_2, \dots, x_{n-1}, x_n) = a_1^{\pm} b_1^{\mp} + a_2^{\pm} b_2^{\mp} + \dots + a_k^{\pm} b_k^{\mp}$ that can be expressed as an OR sum of symmetric periodic subfunctions, $F_s = \cup_{i=1}^l F_p^i(x)$, where each $F_p^i(x)$ is a periodic symmetric function of period T_i and with two transitions $a_i \in \{a_j\}_{j=1,k}$ and $b_i \in \{b_j\}_{j=1,k}$ inside the first period, can be implemented with a feedforward network with both size and depth, measured in terms of neural gates, in the order of $O(\log n)$.

Proof: Each periodic subfunction $F_p^i(x)$, defined by the triplet $\{a_i, b_i, T_i\}$, can be implemented with a feedforward network with $1 + \lceil \log(\lceil (n - a_i)/T_i \rceil + 1) \rceil$ neural gates. Because F_s is true anytime one subfunction $F_p^i(x)$ is true one neural gate is enough to implement the summation of the subfunctions. This leads to an overall implementation cost $C(F_s) = \sum_{i=1}^l (1 + \lceil \log(\lceil (n - a_i)/T_i \rceil + 1) \rceil) + 1$ neural gates. From this cost we can derive an asymptotic bound as follows:

$$\begin{aligned} C(F_s) &= \sum_{i=1}^l \left(1 + \left\lceil \log \left(\left\lceil \frac{n - a_i}{T_i} \right\rceil + 1 \right) \right\rceil \right) \\ &+ 1 \leq \sum_{i=1}^l \left(3 + \log \frac{n - a_i}{T_i} \right) + 1 \\ &= 3l + 1 + \sum_{i=1}^l \log \frac{n - a_i}{T_i} \\ &= 3l + 1 + \log \frac{(n - a_{i_1})(n - a_{i_2}) \dots (n - a_{i_l})}{T_{i_1} T_{i_2} \dots T_{i_l}} \\ &\leq 3l + 1 + \log \left(\frac{n - a_{i_k}}{T_{i_k}} \right)^l \\ &= 3l + 1 + l(n - a_{i_k}) - l \log T_{i_k} \\ &\leq 3l + 1 + l \log(n - a_{i_k}) \leq l \log n + 3l + 1 \end{aligned}$$

where $a_{i_k} = \min\{a_i\}_{i=1,l}$, $T_{i_k} = \min\{T_i\}_{i=1,l}$ and it was considered that in the worst case scenario T_{i_k} is equal to two. Given that we assumed that the number of harmonics l is a constant the previous result provides an asymptotic complexity in the order of $O(\log n)$ for the network size. The depth of the network is also in order of $O(\log n)$ because it is upper bounded by $2 + \lceil \log(\lceil (n - a_{i_k})/T_{i_k} \rceil + 1) \rceil$. ■

V. SERIAL BINARY ADDITION

Generally speaking, the binary addition of two n -bit operands is performed by adding two operands of length n into a single $(n + 1)$ -bit number representing the sum. Given that the maximum value of an n -bit number enumerated from $n - 1$ to zero, with the bit enumerated by $n - 1$ being the MSB, is $2^n - 1$ then the binary addition can assume up to $2(2^n - 1) + 1$ different distinct output values. This is equivalent to producing a counter capable of operating on weighted inputs and counting up to the value $2^{n+1} - 1$.

An $n|r$ counter determines how many of its inputs are "1" and express this result as a binary number at its output [24]. In order to be able to represent binary numbers up to n the number of counter's outputs r has to be equal to $1 + \lceil \log n \rceil$. Based on the observation that the LSB output bit of such a counter is actually an n -variable parity function computed over all the n input bits and on the fact that as a consequence of Remark 1 the feedforward network depicted in Fig. 5 can be used for the implementation of the parity function of n variables with $1 + \lceil \log n \rceil$ neural gates, we prove that the entire $n|(1 + \lceil \log n \rceil)$ counter can be implemented with feedforward neural networks at the same cost as the n -variable parity function.

Theorem 4: Any $n|(1 + \lceil \log n \rceil)$ counter can be implemented with an $1 + \lceil \log n \rceil$ depth and size feedforward neural network.

Proof: By its definition, following the base 2 counting rules, any counter's output bit $S_i, i = 0, 1, \dots, \lceil \log n \rceil$ is equal to one inside an interval that includes 2^i consecutive integers, every 2^{i+1} integers, and zero otherwise. Thus each bit S_i can be described by a periodic symmetric function with period 2^{i+1} . This is equivalent to the fact that each bit has its value given by a periodic symmetric function with period 2^{i+1} and two transitions inside each interval. We proved in Theorem 1 that such periodic symmetric functions can be implemented in logarithmic delay and cost with feedforward networks.

In Fig. 7 is graphically depicted the network implementing the periodic symmetric function corresponding to S_0 , i.e., the LSB of the counter. As it can be observed in the figure the neural gate on level $r - 2$ provides at its output the value of a periodic symmetric function with period 2^2 , i.e., value that correspond to the bit S_1 , the neural gate on level $r - 3$ provides at its output the value of a periodic symmetric function with period 2^4 that gives exactly the value of the bit S_2 and so forth an so on. By decreasing the level of the gate with one the period of the generated function double its value. The proof of this result follow straight-forward from Theorem 1 and the definition of the counter. Therefore each gate in the network is producing an output bit of the counter. The fact that each gate produce a certain bit, after a delay that is given by the level of the gate in the network, allows the sharing of the gates and leads to the implementation of all the $1 + \lceil \log n \rceil$ functions that give the values of the output bits of the counter only with $1 + \lceil \log n \rceil$ neural gates. ■

Note that even if the global delay is $1 + \lceil \log n \rceil$ each bit S_i takes the valid value after a delay that is in inverse relation with its significance. Therefore this solution provides,

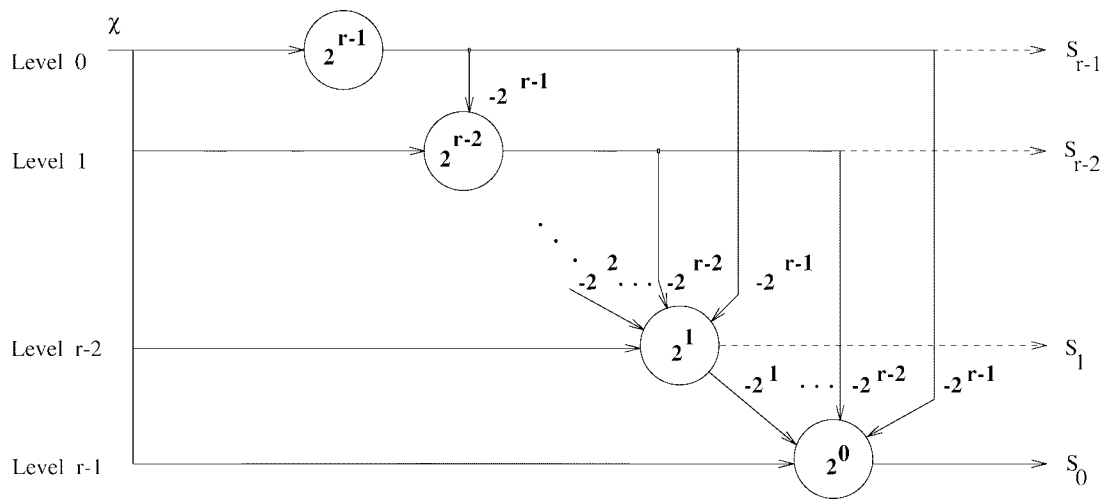


Fig. 7. $n|r$ Counter feedforward neural network implementation.

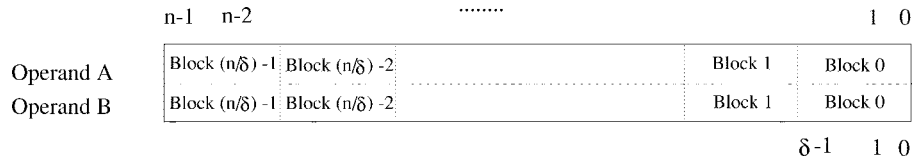


Fig. 8. Partition of operands in (n/δ) δ -bit blocks.

because of the structure of the network, first the MSB (carry-out) of the counter and at the end the LSB of the counter. This is an interesting and useful peculiarity of the counters that are implemented with threshold networks constructed with the method we introduced in the previous theorem.

In general if the implementation restrictions for the weight values are neglected it can be stated that such a $2(2^n - 1)(n + 1)$ counter is able to perform the addition of two n -bit numbers. The counter can be built with $n + 1$ gates and each input bit enumerated by $i, i = 0, 1, \dots, n - 1$ has to be weighted¹⁰ with 2^i instead of one. This type of counter can be viewed as a bit serial adder with the delay of $n + 1$. However, this direct solution is rather expensive in terms of delay and network size. Apart of that it is not implementable for realistic operand dimensions because it requires exponentially large (the maximum weight value for n -bit operands is 2^n) weight values. In the following we present a new scheme for serial addition that substantially improve all of the design parameters.

Assume that the operands are applied in a δ -bit¹¹ serial fashion, δ being an integer greater than or equal to one. This is equivalent with the partition of the two n -bit operands A and B into $\lceil n/\delta \rceil$ blocks of δ bits, as depicted in Fig. 8, and assume that the data arrive at the rate of δ -bits per clock period. At most one block, the last block, can have less than δ bits in the case that n is not divisible by δ . For simplicity of notation we assume that the result of the division n/δ is an integer number.

¹⁰It is also possible to replicate the inputs instead of giving them weights larger than one.

¹¹We note here that the δ notation is not casual. We refer here to δ because as it will verified later this notation suggests a clear link between the delay of the schemes we proposed and δ -bit computation.

If the block pairs are enumerated from $(n/\delta - 1)$ to zero, with the least significant pair enumerated by zero, the sum bits that correspond to pair i can be computed independently if we know the carry-in into the block i , i.e., the carry out that results from the summation of the pair $i - 1$. A feedforward network with $\delta + 1$ neural gates produces the sum bits that correspond to the addition of a pair of blocks. This is because the binary addition of two δ -bit numbers can assume up to $2(2^\delta - 1) + 1$ different distinct values and these values can be produced with a $2(2^\delta - 1)(\delta + 1)$ counter. Given that the implementation of this counter with neural gates is able to provide the carry-out of a block after the delay of one neural gate we can consider one pair of blocks, i.e., 2δ bits, each clock period. The period of the clock is imposed by the maximum delay of the slowest neural gate in the network. In order to be able to operate in a pipeline environment we have to modify the structure of the counter by adding intermediate pipeline registers between the network's levels. We have also to add a feedback line from the output of the first gate to its input register in order to provide the valid carry-in for the next pair of blocks. Note that for a counter that adds two blocks of δ bits the input bit in the position $i, i = 0, 1, \dots, \delta - 1$, to avoid replication, requires a weight value of 2^i .

Given that addition can be viewed as a counting function and because the network we proposed in Fig. 7 can be trivially modified to compute pipelined counting we can perform binary addition in a δ -serial¹² fashion with a pipeline structure constructed around a neural gate-based feedforward network with $\delta + 1$ gates and pipeline stages. Given this global

¹²That is to say that both operands are coming with a transmission rate of δ bits per serial cycle, with a serial cycle comprising a threshold gate and a latch.

scheme we will investigate in the remainder of this section the following questions.

- Assuming a transmission rate of δ -bit per cycle which are the costs and the delay performance of a feedforward neural network that implements the serial addition of two n -bit operands;¹³

- What are the consequences of polynomially bounded weight values on the delay, cost, and fan-in?

Theorem 5: Assuming that serial addition is performed with the partitioning of the two operands in $\lceil n/\delta \rceil$ blocks of at most δ bits and that it is implemented with a feedforward neural network the overall delay is $\delta + n/\delta$ serial cycles. The implementation cost, in terms of neural gates is $\delta + 1$ and $\frac{1}{2}(5\delta^2 + 9\delta) + 2$, in terms of latches. The maximum weight value is 2^δ and the maximum fan-in is $3\delta + 1$.

Proof: Because we partition the operands in $\lceil n/\delta \rceil$ blocks, each block containing at most δ bits the maximum sum for each block is $2(2^\delta - 1)$. Therefore the sum bits for a pair of blocks can be computed by a $\delta + 1$ stages feedforward network and the overall delay Δ of the proposed structure is given by

$$\Delta = \delta + 1 + \frac{n - \delta}{\delta} = \delta + \frac{n}{\delta}. \quad (2)$$

The structure has $2\delta + 1$ latches at the input in order to be able to memorize the two blocks and the global carry-in, $2\delta + 1 + i$ latches in order to transfer the data between the levels $i - 1$ and i , for $i = 1, 2, \dots, \delta$ and $\delta + 1$ latches at the output in order to store the output sum bits. Therefore the overall number of latches is given by

$$\begin{aligned} R &= 2\delta + 1 + \sum_{i=1}^{\delta} (2\delta + 1 + i) + \delta + 1 \\ &= 3\delta + 2 + 2\delta \sum_{i=1}^{\delta} 1 + \sum_{i=1}^{\delta} i + \sum_{i=1}^{\delta} 1 \\ &= 2\delta^2 + 4\delta + 2 + \frac{1}{2} \delta(\delta + 1) = \frac{1}{2} (5\delta^2 + 9\delta) + 2. \quad (3) \end{aligned}$$

For this type of feedforward networks the maximum weight value is 2^δ . The gate with the maximum fan-in is the gate on the level δ of the feedforward network. For this gate the fan-in is given by the number of bits that participate in one computation step, i.e., $2\delta + 1$, plus the number of gates that are above this gate in the feedforward network, i.e., δ . This leads to a maximum fan-in of $3\delta + 1$. ■

As a consequence of the δ partition the gates in the feedforward network have weights with exponential size and large fan-in. However in practice neural gates with large weights and fan-in values can not be implemented [3], [4]. Because of this we are interested in solutions that fulfill the small weights and fan-in requirements and that do not deteriorate substantially the delay. Consequently, we introduce a theorem that investigate the delay, cost and fan-in of the feedforward neural gates networks that perform serial addition under the assumption that the weight values are polynomially bounded.

¹³We assume the single addition issue and we develop fully pipelined schemes and therefore the performance we report will improve for back-to-back additions.

Theorem 6: The serial addition of two n -bit operands can be computed in $O(n/\log n)$ serial cycles by a feedforward neural network with polynomially bounded weight values and a maximum fan-in in the order of $O(\log n)$. The implementation cost is in the order of $O(\log n)$, in terms of neural gates, and in the order of $O(\log^2 n)$, in terms of latches.

Proof: Partition the operands into blocks of at most $\delta = k \log n$ bits, with k an integer constant. First implication here is that the data transmission rate is $k \log n$ bits per cycle, dictating the most bits a polynomially bounded weights implementation requires. Because the maximum weight is given by 2^δ this partition choice will lead to a maximum weight value of $2^{k \log n} = n^k$, i.e., to polynomially bounded weight values.

The overall delay for the addition in serial cycles is given by $k \log n + (n/k \log n)$, as a consequence of (2) and this is in the order of $O(n/\log n)$. The cost of the feedforward network, in terms of neural gates, is given by $k \log n + 1$ which is indeed in the order of $O(\log n)$. The cost of the feedforward network, in terms of latches, is given, as a consequence of (3), by $\frac{1}{2}(5(k \log n)^2 + 9k \log n) + 2$ which is in the order of $O(\log^2 n)$. The maximum fan-in is given by $3k \log n + 1$ and this is in the order of $O(\log n)$. ■

VI. SERIAL BINARY MULTIPLICATION

In this section we propose a logarithmic depth and size feedforward neural network for serial multiplication. We assume the multiplication of two n -bit operands and a rectangular¹⁴ $2n \times n$ multioperand matrix [8] and discuss networks capable of reducing the multioperand matrix directly into the final sum in a block serial manner.

Theorem 7: The serial multiplication of two n -bit operands can be computed by a $O(\log n)$ depth and size, both measured in terms of neural gates, feedforward network in $O(n)$ serial cycles. The implementation cost in terms of latches is in the order of $O(n \log n)$, the maximum weight value is in the order of $O(n^2)$ and the maximum fan-in is in the order of $O(n \log n)$.

Proof: We divide the multioperand matrix into blocks each block containing $\log n$ columns and n rows. Under this assumption the sum corresponding to a block can be at most $n(2^{\log n} - 1)$ and it can be represented on $2 \log n$ bits. This sum can be computed by a $n(n - 1) \lfloor 2 \log n \rfloor$ counter and as we proved in Theorem 4 such a counter can be implemented by a neural network with $2 \log n$ depth and size.

Let assume that the blocks are $B_j, j = 0, 1, \dots, \lceil 2n/\log n \rceil - 1$ and the output bits produced by the counter are $P_i, i = 0, 1, \dots, 2 \log n - 1$ with the bit enumerated as zero being the LSB and the bit enumerated as $2 \log n - 1$ the MSB. For any block $B_j, j = 0, 1, \dots, \lceil 2n/\log n \rceil - 2$ the bits $P_i, i = 0, 1, \dots, \log n - 1$ resulting from its summation would not further influence the computation for the block B_{j+1} and can be reported for output. The bits $P_i, i = \log n, \log n + 1, \dots, 2 \log n - 1$ are carries into

¹⁴Actually on each row only n partial products can have values different than zero.

the next block. Therefore, in order to compute the sum for the block B_{j+1} we have to consider $n + 1$ rows instead of n . However this will not change the depth and the cost of the counter we use because in this case the maximum sum can be $(n + 1)(2^{\log n} - 1) = n^2 - 1$ and this value can be still represented on $2\log n$ bits.

In order to be able to operate in a pipeline environment we have to modify the structure of the counter by adding intermediate pipeline registers between the levels. We have also to add feedback lines from the output bits enumerated as $\log n, \log n + 1, \dots, 2\log n - 1$ to the input register in order to provide the valid carries-in for the next block. Given that the neural network we consider for the implementation of the $(n^2 - 1)2\log n$ counter is able to provide the MSB's first we can consider one new block after all the carries are ready, i.e., after $\log n$ serial cycles, with the serial cycle comprising a neural gate and a pipeline latch. Under these assumptions the entire multiplication can be performed in $\Delta = \log n(2n/\log n) + \log n = 2n + \log n$ serial cycles and this is in the order of $O(n)$.

As long as our scheme uses only one $(n^2 - 1)2\log n$ counter the cost of the network measured in terms of neural gates is $2\log n$. The structure has $(n + 1)\log n$ latches at the input in order to be able to memorize the block bits and the carries-in, $(n + 1)\log n + i$ latches in order to transfer the data between the levels $i - 1$ and i , for $i = 1, 2, \dots, 2\log n - 1$ and $2\log n$ latches at the output in order to store the output sum bits. Therefore the overall number of latches is given by

$$\begin{aligned} R &= (n + 1)\log n + \sum_{i=1}^{2\log n - 1} ((n + 1)\log n + i) \\ &\quad + 2\log n \\ &= 2(n + 2)\log n + \sum_{i=1}^{2\log n - 1} i = 2(n + 2)\log n \\ &\quad + \frac{1}{2} 2\log n(2\log n - 1) \end{aligned}$$

and this value is in the order of $O(n \log n)$.

For this type of feedforward networks the maximum weight value is given by $2^{2\log n} = n^2$.

The gate with the maximum fan-in is the gate on the level $2\log n$ of the feedforward network. For this gate the fan-in is given by the number of bits that participate in one computation step, i.e., $(n + 1)\log n$, plus the number of gates that are above this gate in the feedforward network, i.e., $2\log n$. This leads to a maximum fan-in of $(n + 3)\log n$ which is in the order of $O(n \log n)$. ■

VII. CONCLUSIONS

In this paper we investigated the realization of a general class of Boolean symmetric functions, the periodic symmetric Boolean functions. We have shown that: any n -input periodical symmetric function F_p with period T , two transitions inside a period, and the first positive transition at $x = a$ can be implemented with an $1 + \lceil \log(\lceil (n - a)/T \rceil + 1) \rceil$ depth and size network, both measured in terms of neurons; any n -input periodical symmetric function with period T , more than two

transitions inside a period, and the first positive transition at $x = a_1$ can be implemented with a $t + \lceil \log(\lceil (n - a_1)/T \rceil + 1) \rceil$ depth and size network, where t is the number of neurons necessary to implement the restriction of F_p to the first period. We derived an asymptotic bound of $O(\log n)$ for the neural realization size and depth for a larger class of symmetric functions (which includes periodic symmetric functions), namely the symmetric functions that can be decomposed into a constant number of periodic symmetric subfunctions. The most general problem, the realization of generic symmetric Boolean functions, possibly by a $O(\log n)$ depth and size network still remains open and the subject of future research.

Given that a number of functions performing computations in computers belong to the class of functions we considered, or can be generalized to these functions, we investigated the benefit our results can have if used in the implementation of serial binary addition and multiplication. We assumed feedforward neural networks without learning and with polynomially bounded weights, and we proposed new schemes for serial binary addition and multiplication. An overall delay of $O(n/\log n)$ serial cycles, with a serial cycle comprising a neural gate and a pipeline latch, was derived for the serial addition of two n -bit operands. The implementation cost is in the order of $O(\log n)$, in terms of neurons, and in the order of $O(\log^2 n)$, in terms of latches. The weight values are polynomially bounded and the maximum fan-in is in the order of $O(\log n)$. Concerning the serial multiplication of two n -bit operands we proved that it can be computed in $O(n)$ serial cycles. The implementation cost is in the order of $O(\log n)$, in terms of neurons, and in the order of $O(n \log n)$ in terms of latches. The maximum weight value is in the order of $O(n^2)$ and the maximum fan-in is in the order of $O(n \log n)$.

REFERENCES

- [1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.* 5, pp. 115–133, 1943. Reprinted in "Neurocomputing foundations of research" J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA: MIT Press, 1988.
- [2] W. Pitts and W. S. McCulloch, "How we know universals: The perception of auditory and visual forms," *Bull. Math. Biophys.* 9, pp. 127–147, 1947. Reprinted in "Neurocomputing foundations of research" J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA: MIT Press, 1988.
- [3] T. Shibata and T. Ohmi, "Neuron MOS binary-logic integrated circuits—Part I: Design fundamentals for soft-hardware circuit implementation," *IEEE Trans. Electron Devices*, vol. 40, no. 3, pp. 570–575, Mar. 1993.
- [4] T. Shibata and T. Ohmi, "Neuron MOS binary-logic integrated circuits—Part II: Simplifying techniques of circuit configuration and their practical applications," *IEEE Trans. Electron Devices*, vol. 40, pp. 974–979, May 1993.
- [5] H. Ozdemir, A. Kepkep, B. Pamir, Y. Leblebici, and U. Cilingiroglu, "A capacitive threshold-logic gate," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1141–1150, Aug. 1996.
- [6] S. Muroga, *Threshold Logic and its Applications*. New York: Wiley, 1971.
- [7] N. Pippenger, "The complexity of computations by networks," *IBM J. Res. Develop.*, vol. 31, no. 2, pp. 235–343, Mar. 1987.
- [8] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*. New York: Holt, Reinhart, and Winston, 1982.
- [9] S. Muroga, "The principle of majority decision elements and the complexity of their circuits," in *Proc. Int. Conf. Inform. Processing*, June 1959, pp. 400–407.
- [10] R. C. Minnick, "Linear-input logic," *IRE Trans. Electronic Comput.*, vol. EC-10, pp. 6–16, Mar. 1961.

- [11] R. Paturi and M. Saks, "On threshold circuits for parity," in *IEEE Symp. Foundations of Comput. Sci.*, Oct. 1990, pp. 397–404.
- [12] K. Y. Siu, V. Roychowdhury, and T. Kailath, "Depth-size tradeoffs for neural computation," *IEEE Trans. Comput.*, vol. 40, Dec. 1991.
- [13] W. H. Kautz, "The realization of symmetric switching functions with linear-input logical elements," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 371–378, Sept. 1961.
- [14] K. Y. Siu and J. Bruck, "Neural computation of arithmetic functions," *Proc. IEEE*, vol. 78, no. 10, pp. 1669–1675, Oct. 1990.
- [15] K. Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister, "Depth efficient neural networks for division and related problems," *IEEE Trans. Inform. Theory*, vol. 39, pp. 946–956, May 1993.
- [16] S. Vassiliadis, S. Coțofană, and J. Hoekstra, "Block save addition with threshold logic," in *IEEE 29th Asilomar Conf. Signals, Syst., Comput.*, Oct. 1995, pp. 575–579.
- [17] S. Vassiliadis, S. Coțofană, and K. Bertels, " $2 - 1$ Addition and related arithmetic operations with threshold logic," *IEEE Trans. Comput.*, vol. 45, pp. 1062–1068, Sept. 1996.
- [18] M. J. Irwin and R. M. Owens, "A case for digit serial VLSI signal processing," *J. VLSI Signal Processing*, no. 1, pp. 321–334, 1990.
- [19] V. G. Oklobdzija and M. D. Ercegovic, "An on-line square root algorithm," *IEEE Trans. Comput.*, vol. C-31, pp. 70–75, Jan. 1982.
- [20] M. D. Ercegovic, "On-line arithmetic: An overview," in *Real-Time Signal Processing VII*, 1984, vol. 495, pp. 86–93, SPIE.
- [21] M. D. Ercegovic and T. Lang, *On-Line Arithmetic: A Design Methodology and Applications*, vol. VLSI Signal Processing, III, ch. 24. New York: IEEE Press, 1988.
- [22] L. Dadda, "On serial-input multipliers for two's-complement numbers," *IEEE Trans. Comput.*, vol. C-38, no. 9, pp. 1341–1345, Sept. 1989.
- [23] J. M. Muller, "On-line Computing: A survey and some new results," in *Algorithms and Parallel VLSI Architectures II*, P. Quinton and Y. Robert, Eds. Amsterdam, The Netherlands: Elsevier, 1992, pp. 261–272.
- [24] L. Dadda, "Composite parallel counters," *IEEE Trans. Comput.*, vol. C-29, pp. 942–946, Oct. 1980.

Sorin Coțofană was born in Mizil, Romania. He received the M.S. degree in computer science from the Polytechnic University of Bucharest, Romania, and the Ph.D. degree in electrical engineering from Delft University of Technology (T.U. Delft), The Netherlands.

He worked for a decade with the Research and Development Institute for Electronic Components (ICCE) in Bucharest. His work experience in ICCE was related to structured design of digital systems, design rule checking of IC's layout, logic, and mixed-mode simulation of electronic circuits, testability analysis, and image processing. He is currently an Assistant Professor in the Electrical Engineering department of Delft University of Technology, The Netherlands. His research interests include computer arithmetic, parallel architectures, embedded systems, neural networks, fuzzy logic, computational geometry, and computer-aided design.

Stamatis Vassiliadis (S'86–SM'92–F'97) was born in Manolates, Samos, Greece. He received the Dr.Eng. degree in electronic engineering and the Ph.D. degree in computer science.

He is a Professor in the Electrical Engineering department of Delft University of Technology (T.U. Delft), The Netherlands. He has also served in the faculties of Cornell University, Ithaca, NY, and the State University of New York (S.U.N.Y.), Binghamton, NY. He worked for a decade with IBM in the Advanced Workstations and Systems Laboratory in Austin TX, the Mid-Hudson Valley laboratory, Poughkeepsie, NY, and the Glendale Laboratory, Endicott, NY. In IBM he was involved in a number of projects regarding computer design, organizations, and architectures and the leadership to advanced research projects. A number of his design and implementation proposals have been implemented in commercially available systems and processors including the IBM 9370 model 60 computer system, the IBM POWER II, the IBM AS/400 Models 400, 500, and 510, Server Models 40S and 50S, the IBM AS/400 Advanced 36, and the IBM S/390 G4 and G5 computer systems. His research interests include computer architecture, embedded systems, hardware design and functional testing of computer systems, parallel processors, computer arithmetic, neural networks, fuzzy logic and systems, and software engineering.

Dr. Vassiliadis is a member of the IEEE Computer Society. For his work he received numerous awards including 23 levels of Publication Achievement Awards, 15 levels of Invention Achievement Awards and an Outstanding Innovation Award for Engineering/Scientific Hardware Design in 1989. Six of his patents have been rated with the highest patent ranking in IBM and in 1990 he was awarded the highest number of patents in IBM.