O(n) Depth-3 Binary Addition

S. Vassiliadis * K. Bertels

E.E. Dept, T.U. Delft, Delft The Netherlands 2628 CD
University of Namur Namur, Belgium

Abstract

In this paper we investigate small depth and size feed-forward neural networks performing binary addition. We propose a set of equations that can be used to realize small depth inexpensive networks for arbitrary operand lengths. In particular we show that O(n) depth-3 networks for the binary addition can be easily constructed having small weight sizes. We also describe a scheme for the design of 32-bit binary adders. When compared to the addition scheme known to produce the least expensive adders in terms of area, using feed-forward neural networks, our scheme requires only 20% of the area in terms of neurons. Consequently our design provides substantial area reduction.

Index terms: Binary Addition, Fed-forward Neural Networks, Majority Gates.

1 Introduction

The feed-forward model for artificial neural networks, also denoted in this presentation as neural networks, is a network of neurons in which there are no closed loops [1]. Formally speaking feed-forward multilayer neural networks, are directed acyclic graphs in which each node is a neuron and each edge connects the output of a neuron to the input of another. The entire network comprises three lists namely:. A list of i input nodes, i.e. FE_j neurons with $1 \leq j \leq i$, a list of m internal nodes, i.e. FE_k neurons with $0 \leq k \leq m$ and a list of r output nodes, i.e. FE_l neurons with $1 \leq l \leq r$. Assuming that an edge constitutes a unit length, the depth of a node, a major influence on the delay of the networks, is defined to be the longest path from the input nodes to that node. The output node having the maximum depth is defined to be the depth of the entire network. The size of the network, indicative of the amount of hardware required to implement a network, is defined to be the number of the functional elements constituting the network. In essence, the size of a network is defined to be the sum $i+m+r$. The McCulloch-Pitts neuron model using threshold gates is commonly used to represent the neurons in feed-forward multlilayer neural networks [1]. In assuming Boolean output functional elements then the McCulloch-Pitts neurons $\mathrm{N} \in N : \{0,1\}^n \longrightarrow \{0,1\}$. In such a model the basic neural element is a linear threshold gate computing a Boolean function $F(X)$ such that:

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

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

Clearly, the McCulloch-Pitts model of the neuron comprises of a set of input variables, $X = (x_1, x_2, \ldots, x_{n-1}, x_n)$, a set of weights $\Omega = (\omega_1, \omega_2, \ldots, \omega_{n-1}, \omega_n)$ associated with the inputs, a threshold value, ψ, a summation device, Σ, computing $\mathcal{F}(X)$ and a threshold element, T, computing $F(X) = sgn(\mathcal{F}(X))$.

It is well known that an arbitrary Boolean function can be computed using AND OR and NOT logical gates and that the McCulloch-Pitts model for Boolean feed-forward networks can compute AND, OR and NOT logic circuits [1]. In this paper we consider high speed neural networks (measured in terms of circuit depth) that require small area (measured in terms of neuron). More in particular we are concentrating in improving the area of depth-3 binary addition known to require $O(n^2)$ [4].

The paper is organized as follows: First we provide some background information helpful to follow the discussion. Consequently we present the main theorems necessary to construct an $O(n)$ depth-3 binary adder. Finally, we discuss the design of 32-bit binary adder and conclude with some remarks.

*email: stamatis@duteca.et.tudelft.nl

2 Depth-3 Binary Addition

We proceed for the computation of the sum by providing a recursive equation for the carry. Let the addition be divided into d consecutive groups, enumerated from 0 to $d-1$ with 0 being the least significant group. Further consider for simplicity that all groups have the same length, l. Consider the i^{th} group and the bit within the group enumerated as k. We note that for a group of bits of length l, there are two possibilities for the carry-out to be equal to 1:

- The carry-out from the group is equal to 1 when the sum of the bits comprising the group exceeds or is equal to the value of 2^l.
- The carry-out is equal to the carry-in the group (thus the carry-out is equal to 1 when the carry-in is equal to 1) when the value of the group sum is equal to $2^l - 1$.

For example, assuming that a group comprises four bits the carry-out of the group, is either equal to 1 when the sum is greater than or equal to $2^4 = 16$ or it can be one if the sum is equal to $2^4 - 1 = 15$ and the carry-in is equal to 1. In order to produce the carry equations, for a group i of length l, we define two new quantities, α_i, (the carry-force quantity) and β_i, (the carry-preserve quantity) defined by the following:

- *carry-force:* $\alpha_i = 1$ when the sum has a value $[2^l]^+_-$ and 0 otherwise.
- *carry-preserve:* $\beta_i = 1$ when the sum has a value $[2^l - 1]^+_-$ and 0 otherwise.

The computation of the $\alpha_i = 1$ and $\beta_i = 1$ for a group i comprising of four bits is reported in Figure 1. It should be noted that for the simplicity of the figure the inputs for the threshold values have not been reported in the figure. It can be postulated that the carry-out of the group i can be computed by the following logical equation: $C_i = \alpha_i + \beta_i C_{i-1}$. Clearly, when $\alpha_i = 1$ [1] $C_i = 1$ thus independent of the the carry-in the carry-out is equal to 1. If $\beta_i = 1$ and α_i is equal to 0 and if the carry is equal to 1 then the carry-out of the group i is also equal to 1. Thus the logical expression will compute the carry-out of the group i from the carry-in. Clearly there is a similarity between the carry-force and carry-preserve quantities proposed here and the familiar generate, G, transmit, T, and the pseudo-generate, G^*, and pseudo-transmit, T^*, signals [7, 3, 5, 2]. While equivalence may exist between the quantities defined here and the well known

[1] Note that in this case also $\beta_i = 1$.

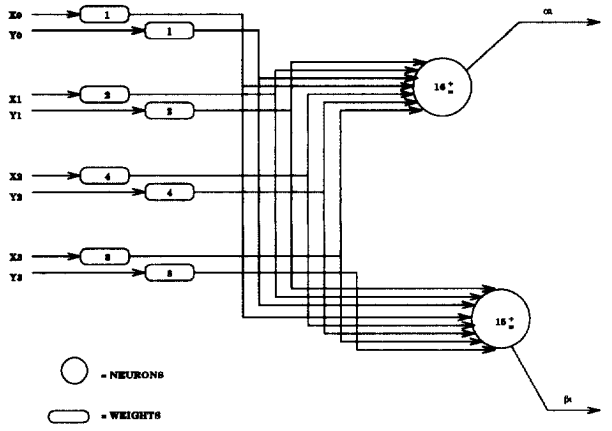

Figure 1: α_i β_i, for a Group i of Four Bits

adder quantities in certain cases [2], in general there is no equivalence between what is defined here and the traditional quantities. For example while it is true that if $\alpha_i = 1$ then also $\beta_i = 1$ for a group say of four bits it is not true that when the group generate signal of a group of four bits is equal to 1 then the transmit signal of the group is not also equal to 1 in general. The expression derived previously provides a logical equation that recursively computes the carry. The following theorems, proven in detail in [6], introduce the threshold equations necessary to compute the sum:

Theorem 1 : For any given group i, the carry-out of the group i, C_i can be computed by: $C_i = sgn\{\gamma_i - 1\}$ with: $\gamma_i = 2^i[\alpha_i + \beta_i - 1] + \gamma_{i-1}$ for $0 \leq i$ and $\gamma_{-1} = C_{in}$

Given that the expressions are not intuitive, we provide some inside to the formulae appearing in the theorem via an example. Assume that we are interested in computing C_2 (the carry-out of the third group). By removing the recursivity, the logical expression determining the carry, using the α and β quantities, is as follows:

$$C_2 = \alpha_2\beta_2 + \beta_2\alpha_1 + \beta_2\beta_1\alpha_0 + \beta_2\beta_1\beta_0 C_{in}$$

The proposed expression for the carry, suggests that:

$$\begin{aligned} C_2 \;=\; & sgn\{2^2(\alpha_2 + \beta_2) + 2^1(\alpha_1 + \beta_1) \\ & + \alpha_0 + \beta_0 + C_{in} - 2^3\} \end{aligned}$$

[2] For example, when we consider the group to be a single bit group comprising bit m then if $G_m = 1$ then $T_m = 1$.

Clearly, independent of the values of the other quantities if $\alpha_2 = 1$ then $\beta_2 = 1$, and $2^2(\alpha_2 + \beta_2) = 2^3$ implying that the carry will be equal to 1. As the logical expression dictates, in case $\alpha_2 = 0$, it must be that $\beta_2 = 1$ for the carry to have a chance of being be 1. The maximum the rest of the expression can assume is when all quantities are equal to 1 thus they can assume the value: $2^1(1+1)+1+1+1 = 7$. Thus it must be that for the carry to be on $\beta_2 = 1$. If $\beta_2 = 1$ and $\alpha_1 = \beta_1 = 1$ then the expression is equal to at least : $2^2 + 22^1 = 2^3$ guaranteeing the carry to be equal to one. Further, the equation also guarantees that the carry is equal to zero when alone $\beta_2 = 1$ or $\beta_1 = 1$ as it is required. By removing necessary recurrences it can also be proven [6] that:

Theorem 2 : Assuming that the carry-in into the addition is C_{in}, the carry-out of the group i, C_i, can be computed by: $C_i = sgn\{\sum_{m=0}^{i} 2^m(\alpha_m + \beta_m) + C_{in} - 2^{i+1}\}$

Regarding the the sum it can be proven that:

Theorem 3 : The sum of two binary numbers can be computed by a depth-3 neural network with O(n) size by: $S_j = sgn\{1_{=}^{+} + 1_{=}^{-} + 3_{=}^{+} - 2\}$ with:

$$1_{=}^{+} = sgn\{2^{i+1}(X_j + Y_j) + \sum_{k=0}^{i} 2^k(\alpha_k + \beta_k) + C_{in} - 2^{i+1}\}.$$

$$1_{=}^{-} = sgn\{2^{i+1} - (2^{i+1}(X_j + Y_j) + \sum_{k=0}^{i} 2^k(\alpha_k + \beta_k) + C_{in})\}.$$

$$3_{=}^{+} = sgn\{2^{i+1}(X_j + Y_j) + \sum_{k=0}^{i} 2^k(\alpha_k + \beta_k) + C_{in} - 3 \cdot 2^{i+1}\}$$

Where the α_k and β_k are computed for all k, except for $k = i$, using the entire group of bits and for $k = i$ the quantities α_k and β_k are computed by considering the bits r of the group i where $0 \leq r \leq j - 1$.

3 The Design of a 32-bit Binary Adder

In designing a 32-bi adder using the theorems presented in the previous section we use the steps of the following general algorithm:

step 1: Subdivide the addition into groups.

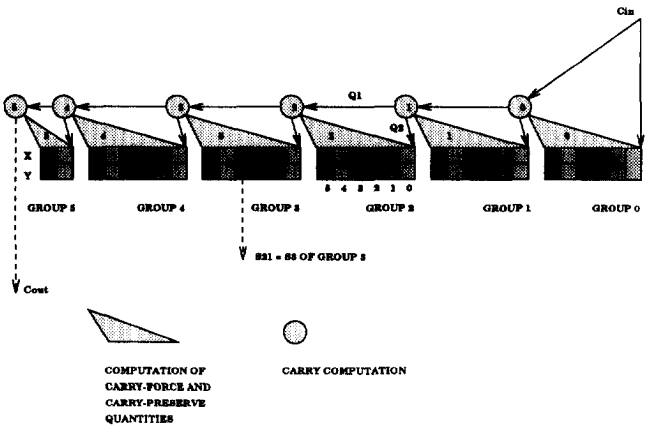

Figure 2: 32-Bit 2-1 Unsigned Binary Adder

step 2: First compute the α and β quantities and then compute the carry into the groups.

step 3(optional): Compute the carry into the bit positions.

step 4: Compute the sum for every bit position using the operand bits and the carry into the groups (if the optional step is not taken) or the carry into the bit positions (if the optional step has been taken).

This procedure is described in Figure 2 where a 32 bit addition is described. The addition is partitioned into six groups, enumerated from 0 to 5. The carry block in the Figure receive quantities of previous groups to produce the carries for every block (the signal Q2 in the Figure), and the auxiliary quantity(the signal Q1 in the Figure) While this process may appear to be sequential (first produce the carry for block r then for block r+1) such a restriction clearly is not necessary.

To clarify the operation of the proposed scheme consider the design for the sum of the bit enumerated over all as 21 (or as bit 3 in group 3).

First Level: [3]

$\alpha_0 = sgn\{\sum_{r=0}^{5} 2^r(X_r + Y_r) - 2^6\}$

[3](NOTE 1: The X_r, Y_r belongs to appropriate groups. For example for the computation of α_0, X_r, Y_r are the operand bits from the first group.) (NOTE 2: To compute the sum for the bit 21, the group enumerated as 3 contains only bits enumerated as 0,1,2, and 3, implying that the computation of the α_3 and β_3 considers only the bits 0,1 and 2 which correspond to bits 18, 19, and 20 of the addition.)

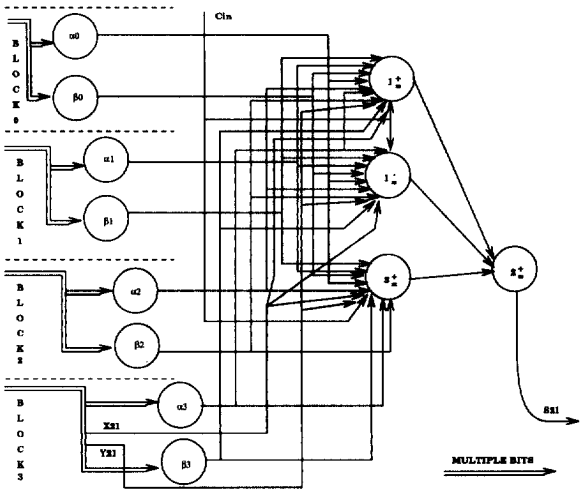

Figure 3: Depth-3 Binary Addition

$$\beta_0 = sgn\{\sum_{r=0}^{5} 2^r(X_r + Y_r) - (2^6 - 1)\}$$
$$\alpha_1 = sgn\{\sum_{r=0}^{5} 2^r(X_r + Y_r) - 2^6\}$$
$$\beta_1 = sgn\{\sum_{r=0}^{5} 2^r(X_r + Y_r) - (2^6 - 1)\}$$
$$\alpha_2 = sgn\{\sum_{r=0}^{5} 2^r(X_r + Y_r) - 2^6\}$$
$$\beta_2 = sgn\{\sum_{r=0}^{5} 2^r(X_r + Y_r) - (2^6 - 1)\}$$
$$\alpha_3 = sgn\{\sum_{r=0}^{2} 2^r(X_r + Y_r) - 2^3\}$$
$$\beta_3 = sgn\{\sum_{r=0}^{2} 2^r(X_r + Y_r) - (2^3 - 1)\}$$

Second Level:

$$1_{=}^{+} = sgn\{2^4(X_{21} + Y_{21}) + \sum_{k=0}^{3} 2^k(\alpha_k + \beta_k) + C_{in} - 2^4\}$$

$$1_{=}^{-} = sgn\{2^4 - (2^4(X_{21} + Y_{21}) + \sum_{k=0}^{3} 2^k(\alpha_k + \beta_k) + C_{in})\}$$

$$3_{=}^{+} = sgn\{2^4(X_{21} + Y_{21}) + \sum_{k=0}^{i} 2^k(\alpha_k + \beta_k) + C_{in} - 3 \cdot 2^4\}.$$

Third Level:

$$S_{21} = sgn\left\{1_{=}^{+} + 1_{=}^{-} + 3_{=}^{+} - 2\right\}$$

The block diagram of the depth-3 sum of bit 21 is shown in Figure 3.

4 Concluding Remarks

The main concern of this paper was the reduction of the size of networks computing binary addition depths with unbounded and bounded weights. We proposed a new set of equations for the addition that allow the 2-1 binary n-bit addition to be computed in a depth-3 network of optimal asymptotic size $O(n)$ with maximum fan-in less than 2n and bounded weights, improving the size requirements of the network for addition of the Siu et al proposal [4]. Our proposal for the example design of a 32 bit binary adder has been estimated to require 204 neurons while the design of the same 32 bit adder using the [4] proposal require 1121 neurons. This we believe constitutes a sizable improvement over what was known to be the least expensive adder design using feed-forward neural networks.

References

[1] I. Aleksander and H. Morton. "An Introduction to Neural Computing". *Chapman and Hall*, 1990.

[2] G. Bewick, P. J. Song, G. De Micheli, and M. J. Flynn. "Approaching a nanosecond: a 32 bit adder,". In *IEEE ICCD 1988*, pages 221–226, Oct. 1988.

[3] H. Ling. "High-speed binary adder,". *IBM J. Res. Develop.*, 25(3):156–166, May 1981.

[4] K.Y. Siu, V. Roychowdhury, and T. Kailath. "Depth-Size Tredeoffs for Neural Computation". *IEEE Transactions on Computers*, Vol. 40, No. 12, December 1991.

[5] S. Vassiliadis. "Recursive equations for hardwired binary adders,". *Int. J. of Electronics*, 67(2):201–213, Aug. 1989.

[6] S. Vassiliadis and K. Bertels. "Feedforward Neural Networks for 2-1 Addition and Related Arithmetic Operations". *T.U. Delft technical report, No 1-68340-44(1994)05, p. 50*, May 1994.

[7] S. Waser and M. J. Flynn. "Introduction to Arithmetic for Digital Systems Designers". *CBS*, 1982.