# Optimum Placement and Routing of Multiplier Partial Product Trees

**Hesham Al-Twaijry and Michael Flynn**

**Technical Report : CSL-TR-96-706**

**September 1996**

This work was supported using facilities supported by NASA contract NAG2-842 and a fellowship from Saudi Arabia.

# Optimum Placement and Routing of Multiplier Partial Product Trees

by

Hesham Al-Twaijry and Michael Flynn

**Technical Report : CSL-TR-96-706**

September 1996

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305-9040
pubs@shasta.stanford.edu

## Abstract

*An algorithm that builds a multiplier under the constraint of a limited number of wiring tracks is designed. The algorithm has been implemented. The program is then used to compare several designs of an IEEE floating point multiplier using several delay models.*

**Key Words and Phrases:** Multiplication, Booth, Trees, Pacement and Routing

## Contents

## List of Figures

## List of Tables

## 1 Introduction

The speed of the multiplier is a critical issue in determining the performance of microprocessors. It has become common for modern microprocessor to have f loating point multipliers implemented fully in hardware, in contrast to in software as in the previous generations. In fact the very latest processors also implement integer multiplication in hardware to speed up address translation, array indexing, and other integer operations.

Multiplication is the process of adding a number of partial products. The different multiplication algorithms differ in how they generate the partial products and how the partial products are added together to produce the final result. The first methods developed where iterative schemes, which are the hardware realization of a simple shift and add algorithm. One of the first departures from the iterative array schemes occured when Wallace [1] introduced the concept of adding the partial products in a carry-free manner using carry save adders, also known as full adders and as (3,2) counters. Wallace showed how to reduce the partial products by connecting the (3,2) counters in parallel in what is known as tree structures. A single carry propagate addition is needed in the final step. Dadda [2] then introduced the term "*parallel (n,m) counter*". These counters are combinational circuits that encode the number of ones that are present in the n input bits using m output bits. Dadda also showed how to minimize the number of counters used in the reduction of the parallel products. The concept of the counter was then extended by Stenzel [3], who showed that the Wallace tree and the Dadda schemes are all special cases of the generalized $(n_{r-1}, n_{r-2}, .., n_1, n_0, m)$ counter. The generalized counter is a counter that encodes several successively weighted input columns and produces their weighted sum.

The tree structures that are described by Wallace and Dadda suffer from irregular interconnections. These irregular connections occur because the counter connections are unique for each partial product weight. There have been several different tree structures that have been proposed by Zuras et al. [4] and Mou et al. [5], that reduce the partial products using more regular interconnections with a slight increase in the number of counter levels over those used by Wallace trees.

A further advance occurred when Weinberger [6] introduced the 4-2 compressor in 1981. The 4-2 compressor is a circuit that compresses 4 inputs of the same arithmetic weight into 2 outputs. In addition there are an internal input and an internal output line. This compressor then was used by Takagi et al. [7] and by Santoro [8] in building binary trees. The binary tree has a more regular structure compared to Wallace trees. The 4-2 compressor was then built upon by Song and De Micheli [9] in the development of the 9-2 compressor family.

The difficulty with the use of the various trees is that not all inputs for the counters have the same delay. Therefore, it is difficult to design a multiplier which takes into account the different delays. The use of the larger counter, such as the 9-2 compressor family, is an attempt to decompose the problem of balancing the different path delays. The use of these larger counters is not without problems because the use of a large counter causes there to be some wasted area.

The best possible solution would be the design of the multiplier using carry-save counters directly with the assistance of software that balances the different paths. This is a global approach that minimizes the total delay for all the possible paths under some specified constraints.

The counter schemes are orthogonal with the use of Booth encoding [10], which is used to reduce the number of partial products. In Booth encoding the number of summands is reduced by recoding the multiplier bits into groups that select multiples of the multiplicand. Higher order Booth encoding reduces the number of summands by a greater degree by using a more complex selection table.

## 2 Previous Work

The design of a multiplier is a complex operation. It is difficult to interconnect the counters to build a design that minimizes the total delay, due to the large number of possible paths and the different path delays from each input. Another complication is that the wire delay is a major contribution to the total delay and that the placement of a counter has an effect on the delay of the outputs of the other counters. The solution to this problem is to have a layout tool that accounts for the different path and wire delays. There have been several algorithms that have been developed to aid the designer. Bewick [11] developed a tool that lays out a multiplier in ECL. It compensates for the extra delays that are caused by the wires by increasing the current to selected gates. This fine tuning is not feasible for CMOS designs because the current drive in CMOS is related to the transistor widths, and varying the transistor widths effects the previous stages, unlike ECL. Another algorithm was developed by Oklobdzija et al, [12] where the design is based upon the carry save counter. This algorithm takes into account the different delays for the paths, in addition to the different input-output delays. However, it ignores the incremental wiring delay that is caused by the placement of the counters. A problem is that these tools connect the counters assuming routing channels with a virtually unlimited number of wiring tracks. In an actual processor design, the counters must be connected using wires that are routed on top of the cells. Therefore, the number of wiring tracks is limited. Therefore, these tools might connect the counters by building trees that require more wires than are available, and therefore the designs obtained can not be implemented.

## 3 Algorithm

The algorithm developed for this paper is based upon Oklobdzija et al, but this algorithm takes into account the input and output delays for the counters and the placement of the counters and their effect on the delay. In contrast Oklobdzija's algorithm places the cells by iteration using information about the wire delays from the previous iteration as a guide.

Another major difference between the algorithms is that the presented algorithm takes into account the limited number of wiring tracks available. Therefore, if there are a large number of wiring tracks available our algorithm produces results that are comparable to those presented by Oklobdzija. On the other hand, if the number of wiring tracks is limited our algorithm still provides an implementable design.

## 3.1 The General Idea

The algorithm connects the fastest possible inputs using a counter. This counter has the effect of removing these fast inputs and replacing them with a slower output and thereby equalizing the delays of the paths in the column and thereby simplifying the choice of which inputs to next connect. The process of connecting the counters is a directed graph problem with each node representing a partial product or a counter and the arc representing the input-output delay for the counters. The problem is thus how to connect the nodes, while minimizing the maximum of the total sum for each possible path.

The algorithm reduces the partial product columns starting with the columns that have the minimum arithmetic weight. The information that is needed to connect the partial products using carry save counters is contained in two lists per column. The first list is the *delay* list $D_i$; this is a list of all the unconnected nodes in the column. This list is used to select the nodes that will be connected and to decide when has the column been fully interconnected. The second list is the *placement* list $P_i$; this is a list of that gives the order in which the nodes are placed and their interconnections. It is also used for the wiring calculations and to update the delay of the nodes, based upon the incremental delays that are caused by the placement of the nodes.

## 3.2 The Algorithm

The presented method first creates two lists $D_i$, $P_i$ for each column of the multiplier. Each element in the delay list $D_i$ is a partial products name, which uniquely identifies the element as a partial product with its column and row location. The other component of each element in $D_i$ is the element's delay. Initially all the partial products are assumed to be available at the same time, and thus this time is set to zero. Each element in the placement list $P_i$ is a partial product that has exactly the same name as those in the delay list and the number of wires that pass over it and its interconnections. Initially the placement lists are sorted so that for each column the partial products for the first rows occur first.

The algorithm then starts with the lists that have the least arithmetic significance, then column i in the partial product array is reduced as follows:

1. If the number of elements in $D_i$ is even and greater than two then a half adder is needed. The half adder is needed because each carry save counter reduces three inputs of weight i and produces a single output of the same weight. That is each counter reduces the number of elements in $D_i$ by two. Therefore to produce a single output *sum* $s_i$ at the end of this process, the number of nodes initially has to be odd. Therefore, a half adder is required if the number of elements in the list is even, so that the number of elements becomes odd. Note that the carry output of the counters has been ignored in this discussion because they are reduced by the column of weight i+1.

2. The half adder is connected to the first two elements in $D_i$ and the two new nodes $h_{i1}$ and $c_{i1}$ are created. the delays for the two new nodes are

$$h_{i1} = MAX(d_{a-h}, d_{b-h},) \qquad (1)$$

$$c_{i1} = MAX(d_{a-c}, d_{b-c},) \qquad (2)$$

For the placement lists the inputs are marked as connected and there names are stored in $h_{i1}$.

3. $h_{i1}$ is inserted in $D_i$ and $P_i$, and $c_{i1}$ is inserted in $D_{i+1}$ and $P_{i+1}$.

4. The two inputs to the half adder are removed from the $D_i$.

5. A counter is selected and connected based upon the following algorithm, as long as the number of elements in $D_i$ is greater than three.

   (a) Sort the elements in $D_i$ in ascending order by the values of the delays then by the names contained in the elements of the list.

   (b) Virtually connect the three inputs with the smallest latency. That is connect the first three elements in the list.

   (c) Based upon the above connection, has the maximum number of wiring tracks been exceeded. This is obtained by calculating the wire track usage up to the place of the last input of the counter in the placement list.

   (d) If the maximum number of wiring tracks has been exceeded then do the following:

      i. If there are more than three unconnected sum and carry outputs that are above the nodes that were virtually connected, connect the fastest three if they have comparable delays.
      If the latency difference between the unconnected wires is large connect the fastest two and a partial product if there are some available otherwise connect the fastest three.

      ii. If there are two or less unconnected sum and carry outputs above the node connect them and some partial products. This creates a linear array.

   (e) Create two new nodes $s_{ij}$ and $c_{ij}$. Calculate the delays for the two new nodes based upon the equation:

$$s_{ij} = MAX(D_a + d_{a-s}, D_b + d_{b-s}, D_c + d_{c-s}) \qquad (3)$$
$$c_{ij} = MAX(D_a + d_{a-c}, D_b + d_{b-c}, D_c + d_{c-c}) \qquad (4)$$

   where $D_i$ represents the delay for the input i and $d_{i-j}$ represents the delay from the input i to output j. The values for $d_{i-j}$ are implementation specific. They are obtained from running HSPICE and they depend on the circuit chosen and the available technology.

   For the placement list $P_i$ mark the inputs as connected and place the connection information in $s_{ij}$.

   (f) Update the delays for the unconnected nodes that occur above the node in the placement list. The nodes are updated to have their delay increased by either the incremental delay that is caused by a counter or by the delay of a counter and the number of partial products the counter has as its inputs. The carry signals have no delay since they have no area in reality. If an input to the counter is a sum its delay is not added because adding its delay means that the incremental delay is added twice.

(g) Insert the sum node $s_{ij}$ in $D_i$ and $P_i$, also insert the carry node $c_{ij}$ into $D_{i+1}$ and $P_{i+1}$ at the correct locations.

(h) Remove the three nodes that have been connected only from the delay list.

The above steps are repeated until there are only three unconnected nodes.

6. When there are three unconnected nodes a counter is used to connect them and the outputs of the counter are connected to the final CPA. They are not inserted in the delay lists.

The above process is then repeated for the next column, until all columns have been processed.

## 4 IEEE Multiplier

The algorithm will be used to design the mantissa part of a floating point multiplier. The multiplier uses the IEEE floating point arithmetic standard [13]. The format for an IEEE double precision number is given in figure 1.

| | | |
|---|---|---|
| Sign Bit | Normalized Fraction | Biased Exponent |
| (1) | (52) | (11) |

Figure 1: Floating Point Number Standard.

The standard defines numbers in a sign-magnitude, normalized format. The standard has a normalized significand, that is the most significant bit of the fraction is always 1, and therefore is not stored. The significand effectively becomes 53 bits. To achieve the rounding accuracy defined by the standard, the full 106 bit result has to be calculated, even though almost half of it is used only for rounding.

We will design the multiplier using three delay models. The delay models increase in realasm. The first model is the simplest model in which the number of counter levels is the criterion used in determining the delay. The second model refines the delay determination by using the number of XORs in the input-output path in the carry-save counter as the delay model. Finally, the last model is a HSPICE simulation of a carry-save counter. The delays used in this model are obtained from the characterization of the circuit. The delays include all parasitic capacitances and resistances. The delays are the worst case delays for each input output pair.

The three delay models are presented to show the advantages of using a more accurate model for the multiplier. The more accurate models show that the minimum delay is acheived using more tracks than what the simpler model states.

## 4.1 CSA Levels

The multipliers designed in this section use the same delay model as that used by Wallace and the subsequent refinements. The delay model uses the same delay from each input to the output and wire delays are ignored. This is a very simplified delay model of limited practical use.

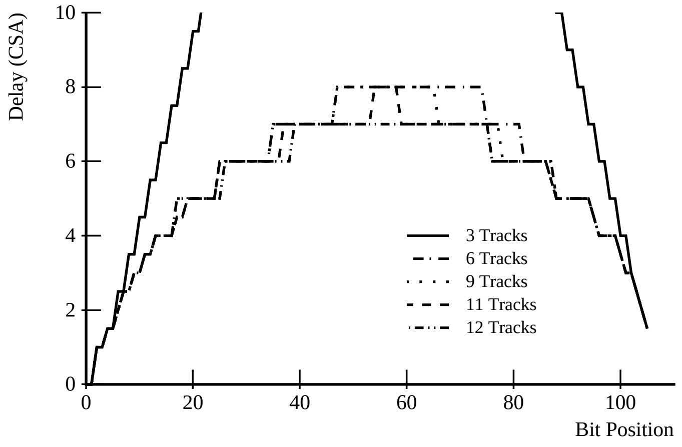

Figure 2: Arrival Times in CSA levels using Booth 2 encoding.

The designs that are compared in figure 2 are all IEEE double precision Booth 2 encoded multipliers. The algorithm gives reasonable result. The algorithm needs a minimum of 12 tracks to be able to give the minimum number of counter levels that are required by Wallace trees as can be seen in figure 2. However by using only half of the tracks there is an increase of only one counter level. The theoretical minimum for the number of tracks needed is 9. This is not achievable by the algorithm.

Figure 2 shows the arrival profile for the input bits for the final carry propagate adder. The arrival profile has a triangular shape. The high order bits arrive before the bits for the middle column. This is surprising since the high order columns depend on the the previous columns. This profile occurs because the last carry from the previous column is an input to the carry propagate adder, and therefore it does not increase the delay for this column. The intermediate carries for the previous column are combined in later stages with the generated sums that have the same delay without any increase in delay.

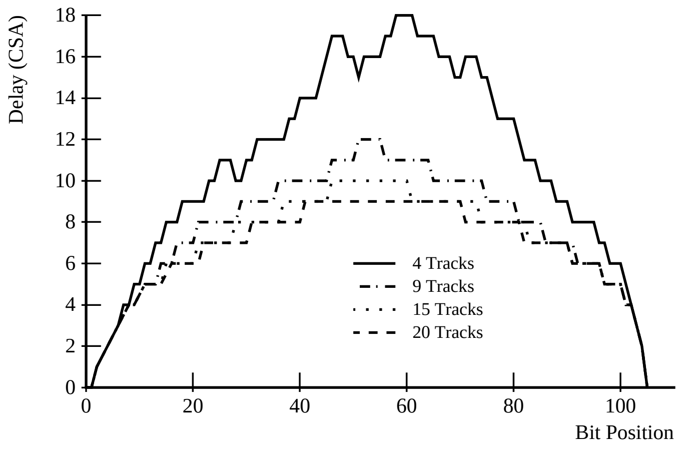

Figure 3: Arrival Times in CSA levels without Booth encoding.

The designs that are compared in figure 3 are IEEE non-Booth encoded multipliers. The minimum number of CSA levels for non-Booth is 9. The minimum number of levels is achieved using 20 tracks. Another point is that increasing the number of wiring tracks by 1 over the linear array to 4 wiring tracks reduces the number of carry save adders from 51 to only 18. Doubling the number of tracks to 9 only reduces the number of counter levels from 6 to 12.

## 4.2 XOR Delays

This section uses the number of XOR gates in the critical path as the basis for comparison. The delays from each input to each output are given in table 1. There are no wiring delays associated with these circuits. These are the same delays as used by Oklobdzija.

| | | Output | |
|---|---|---|---|
| | | Sum (xors) | Carry (xors) |
| HA | A | 1 | 0.5 |
| | B | 1 | 0.5 |
| CSA | A | 2 | 1 |
| | B | 2 | 1 |
| | $C_{in}$ | 1 | 1 |

Table 1: Elements input-output delays in XORS.

The multipliers designed in figure 4 are IEEE double precision multipliers without any Booth encoding. The multipliers designed in this section use a more realistic model for delay than that of the previous section.

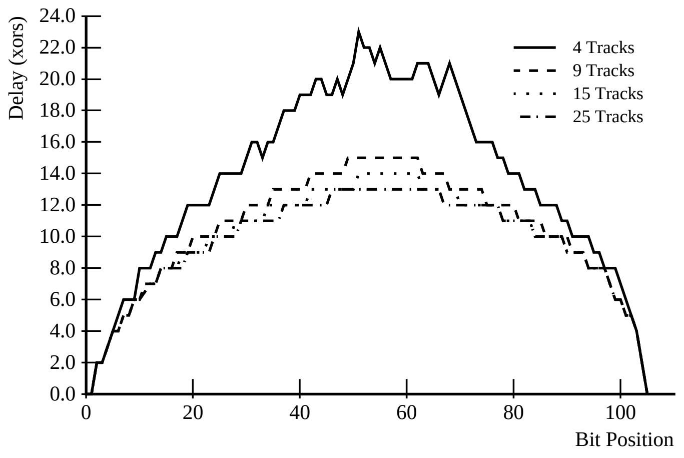

Figure 4: Arrival Times using XOR levels without Booth encoding.

The maximum number of XOR gates in the critical path is 13. This is compared to the number 18 that is obtained from Wallace trees. This shows the advantage of having a delay model that has different delays for each path. The minimum number of XOR levels of 13 is achieved using 25 tracks. Increasing the number of tracks beyond 12 give no improvement in the number of XOR in the critical path until 25 tracks are used. The number of tracks required to achieve the minimum number of XOR levels is larger than that for the number of CSA levels because the nonuniform input-output delays in the carry save adder restrict the outputs of the counters to a smaller subset of possible inputs. However, the actual delays achieved are smaller for the XOR delay model, 14 XOR delays using 12 wiring tracks, compared to the carry save adder delay model, 18 XOR delays using 20 wiring tracks.

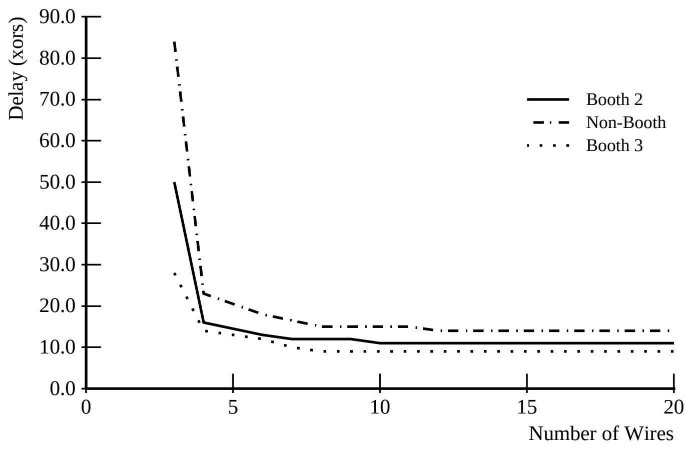

Figure 5: Maximum delay as a function of the number of wires in a channel.

The effect of changing the number of tracks is shown in figure 5. Increasing the number of wires beyond 10 has very limited value. The biggest advantage to increasing the number of wires is for non-Booth encoding. This is expected since non-Booth has the largest number of partial products. Both the Booth 2 and Booth 3 achieve the minimum number of XORs when there are 10 or more tracks. The delays for the larger number of wires are the same because the extra delay caused by the placement of the counters is ignored.

## 4.3 Actual delays

This section uses delays that were obtained using HSPICE and characterizing a DPL carry-save adder [14] that was implemented using $1.0\mu m$ technology. These circuits also included the effects of long wires on the delay. The delays of each input-output pair for the half adder and the carry save adder are given in table 2.

| | | Output | |
|---|---|---|---|
| | | Sum (ns) | Carry (ns) |
| HA | A | 0.717 | 0.558 |
| | B | 0.650 | 0.513 |
| CSA | A | 1.190 | 0.770 |
| | B | 1.185 | 0.780 |
| | $C_{in}$ | 0.779 | 0.472 |

Table 2: Actual Circuit Element Input-Output Delays.

The arrival profiles for the three most common encoding schemes is given in figure 6. The first thing that one notices from the results is that the arrival profile for the bits at the input of the final carry propagate adder is smooth and that it does not contain any steps in contrast to the two previous cases.

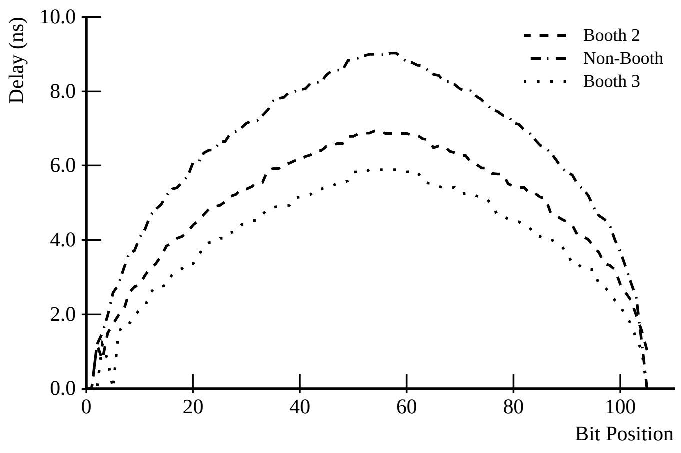

Figure 6: Arrival Times using realistic delays for different encoding schemes.

The profiles in figure 6 are obtained for the case when there are 12 tracks available. The designs ignore the delays associated with the different encoding schemes. For all three cases the maximum delay that is obtained is less that the delay for connecting the minimum number of counters in series using a Wallace tree. For all the cases the minimum number of levels possible has been achieved with the smart interconnection of the counters thereby minimizing the total delay.

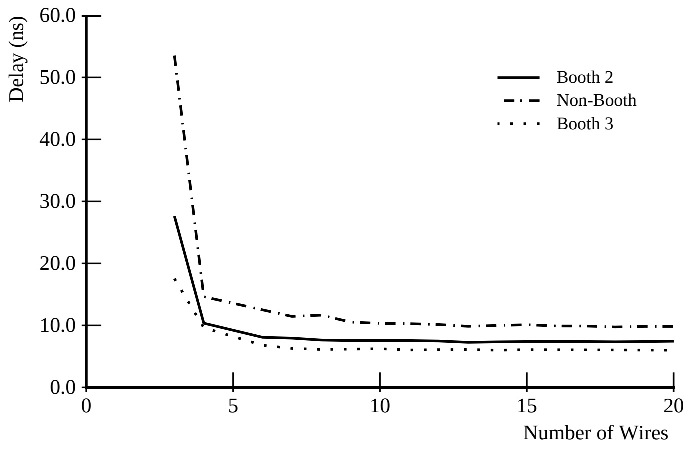

Figure 7: Maximum delay as a function of the number of wires in a channel.

The effect of the number of wires on the total delay for the multiplier is given in figure 7. Increasing the number of wires by 1 beyond that needed by an array to just 4 decreases the total latency by more than half for all three encoding schemes. Increasing the number of wiring tracks beyond 10 is of limited use. It is possible for a design with more wire tracks to have a larger delay than one with less wire tracks. The designs that have more wire tracks have longer wires because the outputs of the carry-save adders are reduced at a later step in the algorithm. Although it should be noted that the differences in delays are minute.

## 5 Conclusion

An algorithm for building multipliers using carry-save counter has been developed. The algorithm is able to build multipliers under the constraints of a limited number of wiring tracks. The algorithm uses a delay model for the elements that has a unique delay for each path through the multiplier. It uses this model to decide on the best possible interconnections for the counter. The algorithm takes into account the incremental delays that are caused by the placement of the counters due to the increase in the wire lengths.

## References

[1] C. Wallace, "A Suggestion for a Fast Multiplier", *IEEE Trans. Electronic Computers*,

pp. 14-17, Feb. 1964.

[2] L. Dadda, "Some Schemes for Parallel Multipliers", *Alta Frequenza*, Vol 34, pp.349-356, Mar 1965

[3] W. Stenzel, "A Compact High-Speed Multiplication Scheme", *IEEE Trans. Computers*, vol C-26, no 10, pp.948-957, Oct 1977

[4] D. Zuras and W. McAllister, "Balanced Delay Trees and Combinatorial Division in VLSI,"*IEEE J. Solid-State Circuits*, vol SC-21, No.5, pp. 814-819, Oct. 1986

[5] Z. Mou and F. Jutand, "A Class of Close to Optimum Adder Trees allowing Regular and Compact Layout", *IEEE Trans. Computers*, pp. 251-254, 1990.

[6] A. Weinberger, "4:2 Carry-Save Adder Module", *IBM Technical Disclosure Bull.*, vol 23, Jan. 1981.

[7] N. Takagi, H. Yasuura, and S. Yajima, "High-Speed VLSI Multilication Algorithm with a Redundant Binary Addition Tree", *IEEE Trans. on Computers*, vol C-34, No. 9, Sept 1985

[8] M. Santoro, "Design and Clocking of VLSI Multipliers", *Ph.D Thesis*, Stanford University, Oct. 1989.

[9] P. Song and G. De Micheli, "Circuit and Architecture Tradeoffs for High-Speed Multiplication." *IEEE J. Solid-State Circuits*, vol SC-26, No.9, pp. 1184-1198, Sep. 1991.

[10] O.L. McSorley, "High Speed Arithmetic in Binary Computers", *Proceedings of the IRE*, 49(1), pp. 67-91, Jan. 1961.

[11] G. Bewick, "Fast Multiplication: Algorithm and Implementations." *Ph.D. Thesis*, Stanford University, Dec. 1993.

[12] V. G. Oklobdzija, D. Villeger and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach" *IEEE Trans. on Computers*, vol C-45, No.3, pp. 294-305, Mar. 1996.

[13] An American National Standard, "IEEE Standard for Floating Point Arithmetic", *ANSI/IEEE standard 754-1985*

[14] N. Ohkubo et al., "A 4.4 ns CMOS 54*54-b Multiplier using Pass-Transistor Multiplexor", *IEEE J. Solid-State Circuits*, vol SC-30, No.3, pp. 251-257, Mar. 1995.