Performance/Area Tradeoffs in Booth Multipliers

Hesham Al-Twaijry and Michael Flynn

Technical Report : CSL-TR-95-684

November 1995

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

Performance/Area Tradeoffs in Booth Multipliers

by

Hesham Al-Twaijry and Michael Flynn

Technical Report : CSL-TR-95-684

November 1995

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305-4055

pubs@shasta.stanford.edu

Abstract

Booth encoding is a method of reducing the number of summands required to produce the multiplication result. This paper compares the performance/area tradeoffs for the different Booth algorithms when trees are used as the summation network. This paper shows that the simple non-Booth algorithm is not a viable design, and that currently Booth 2 is the best design. It also points out that in the future Booth 3 may offer the best performance/area ratio.

Key Words and Phrases: Floating-Point, Multiplication, Booth, Trees

Contents

List of Figures

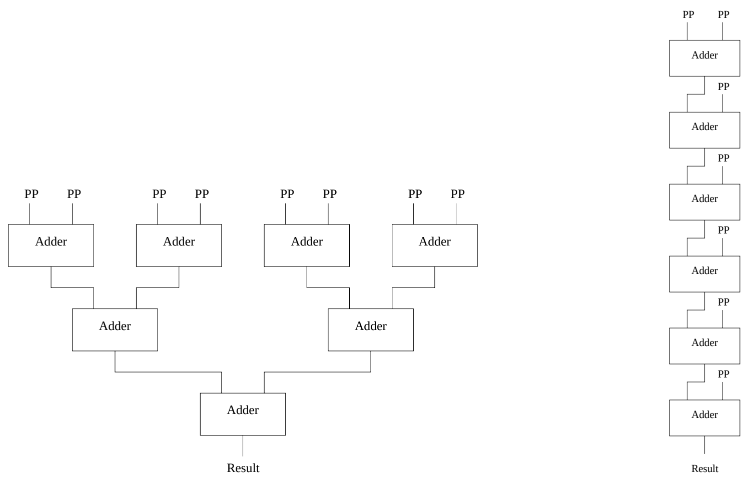


Figure 1: (a) Parallel addition (b)Linear Addition

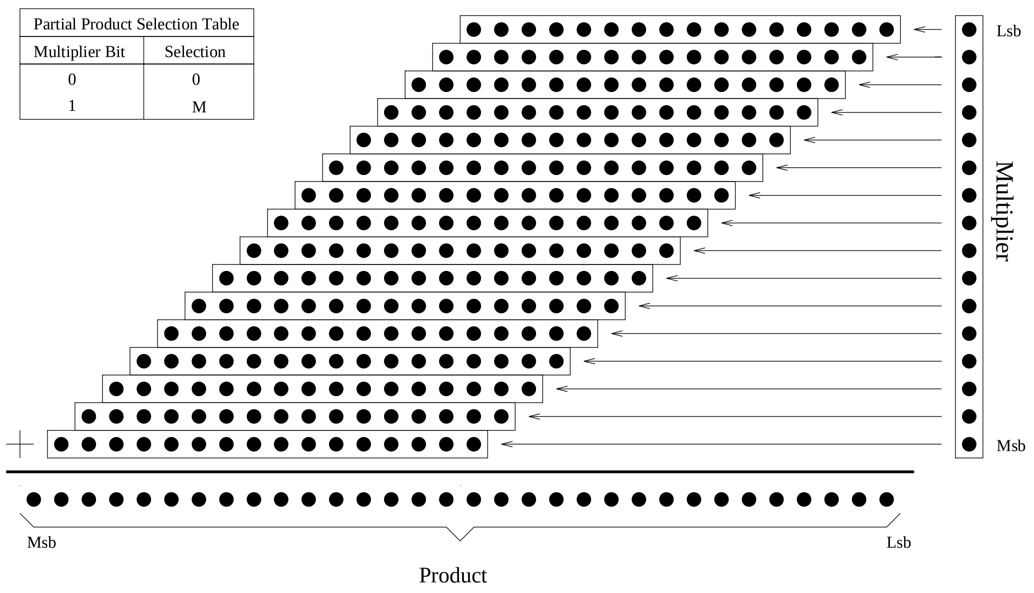
1 Introduction

Multiplication is one of the basic arithmetic operations that constitute programs. In fact 8.72 % of all instructions in typical scientific programs are multiplies [1]. Hardware designers have recognized this and have devoted considerable silicon area to building high speed multipliers.

Multiplication is achieved by the addition of a certain number of summands. Each summand is a chosen multiple of one of the operands (multiplicand), based upon the value of certain bits of the other operand (multiplier). The addition of these summands is a relatively long latency carry propagate addition (CPA). In order to reduce the total time required to produce the result a redundant form of addition, most commonly carry-save addition, is used. In carry-save addition, the summands are split into columns, in which each column's addition progresses independently from adjacent columns. Each column has a certain number of inputs called partial products. In high speed multipliers, the addition of the partial products is done in parallel using tree structures as shown in figure 1(a), in contrast to serially as in linear arrays. The number of adders needed to reduce the partial products is the same for both trees and arrays. The only difference being that trees have more complex interconnections.

The number of summand that must be added to give the multiplications' result can be reduced by using Booth encoding [3]. In Booth encoding the number of summands is reduced by recording the multiplier bits into groups that select multiplies of the multiplicand. Higher order Booth encoding reduces the number of summands by a greater degree by encoding larger groups of multiplier bits and therefore requiring a larger group of multiples to select from and consequently a more complex selection table.

This study investigates the relationship between the topology of the partial product interconnections and the encoding scheme used. It also studies the effect of these topologies and encoding schemes on the latency and area of the multiplier, when the multiplier is part



Partial Product Selection Table	
Multiplier Bit	Selection
0	0
1	M

Figure 2: Non-Booth Encoding

of a larger system.

The remainder of this paper is organized as follows. Section 2 describes the different encoding schemes. In section 3 the topologies that are used are described. Section 4 explains the constraints that were used so that the results can be better understood. Finally, section 5 presents the results, followed by section 6 where the conclusions are given.

2 Encoding Methods

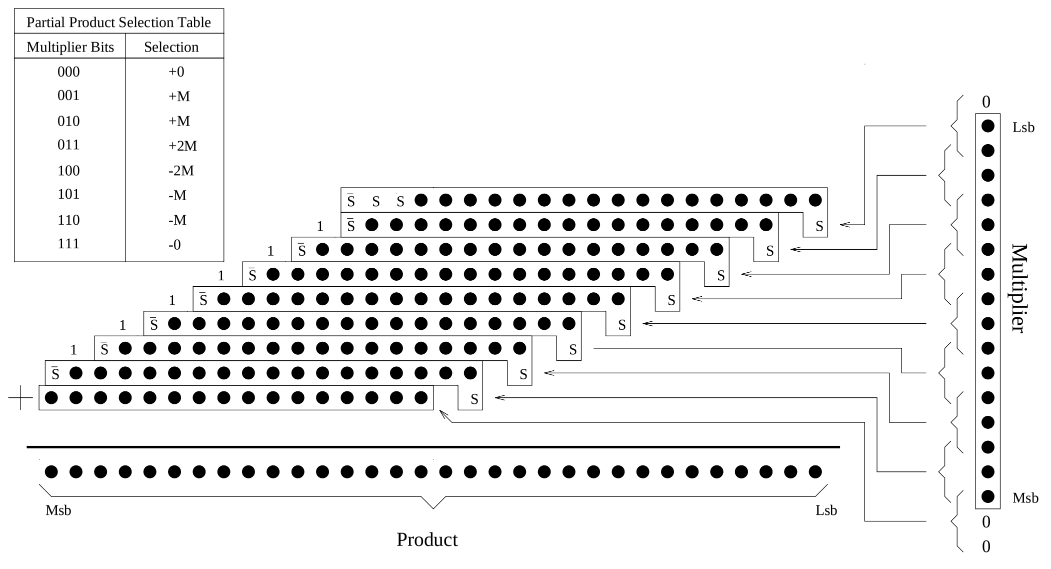
Several methods of encoding the multiplicand are possible. These methods are used to reduce the number of summands that are needed to produce the final result.

2.1 Non-Booth

The first and simplest method for encoding is non-Booth. This algorithm is simply a shift and add algorithm where the multiplicand is conditionally added to produce the final result. In this algorithm the summand is selected from the set $\{0, M\}$. This algorithm's summand selection logic is a simple AND gate. Unfortunately, there is no reduction in the number of multiplicands that need to be summed to produce the final result.

2.2 Booth 2

A smaller number of multiplicand multiples that need to be summed is better. The Booth algorithm attempts to reduce the number of the summands by recording the multiplier so that groups of its bits select multiples of the multiplicand. The Booth algorithm as it was



Partial Product Selection Table	
Multiplier Bits	Selection
000	+0
001	+M
010	+M
011	+2M
100	-2M
101	-M
110	-M
111	-0

Figure 3: Booth 2 Encoding

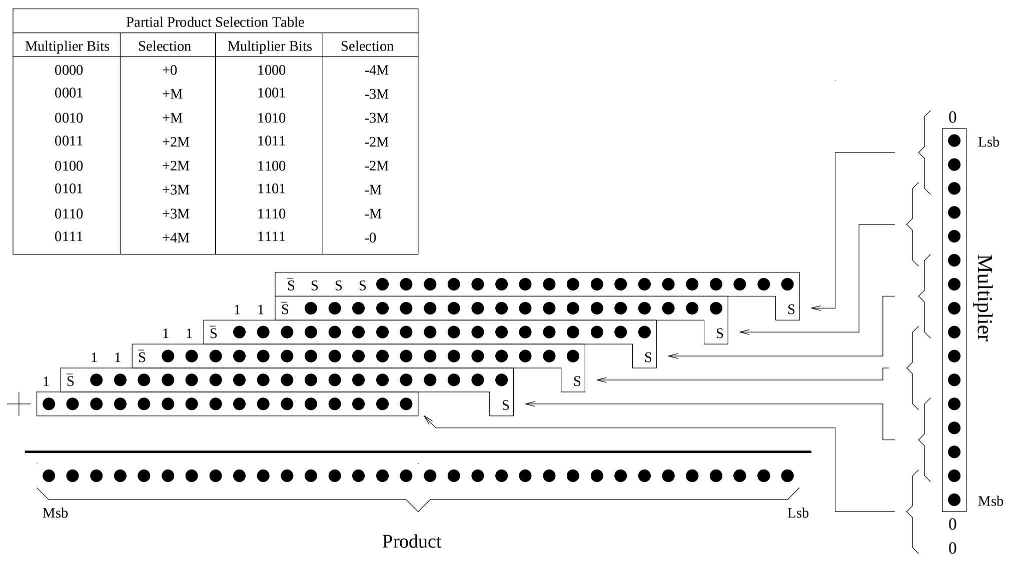
originally proposed performed the encoding serially. Therefore, the Modified Booth Algorithm [3] which performs the encoding in parallel is used. In this algorithm, the multiplier is partitioned into overlapping groups of 3 bits. Each group is decoded in parallel to select a multiple of the multiplicand from the set $\{\pm 2M, \pm M, 0\}$, as shown in figure 3. All of these multiples are obtainable by simple shifting and complementation.

Using this algorithm the number of summands is $\lceil \frac{n+1}{2} \rceil$. The extra 1 in the expression comes from the need to ensure that the last summand is a positive multiple of the multiplicand. This is achieved by adding at least an extra zero to the left of the *multiplier.*

2.3 Booth 3

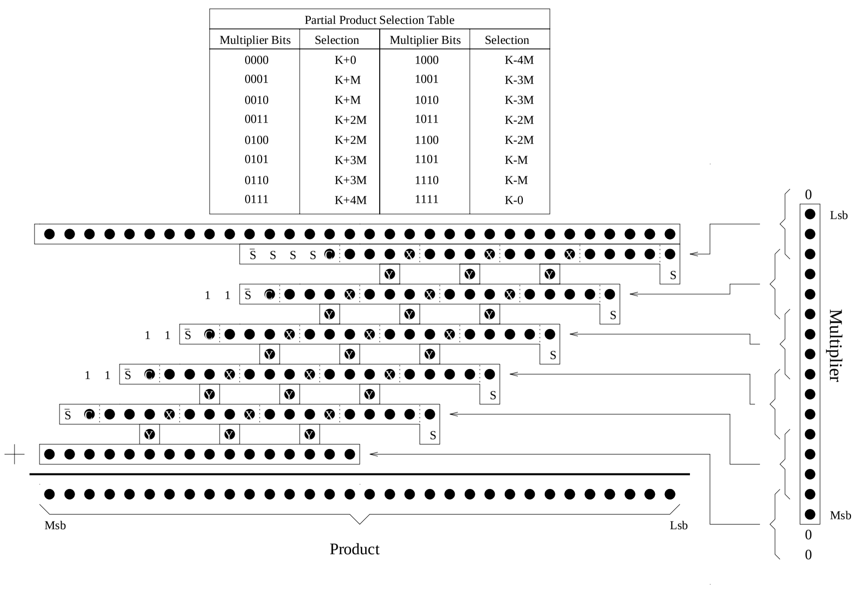
It is also possible to reduce the number of summands required to produce the final result using shift amounts greater than two. In Booth 3 the multiplier is partitioned into overlapping groups of 4 bits, each group is decoded in parallel to select a certain summand. Each summand could be from the set $\{\pm 4M, \pm 3M, \pm 2M, \pm M, 0\}$, as shown in figure 4. All the multiples with except 3M are easily obtainable, by simply shifting and complementing.

The generation of the 3M multiple, which is referred to as a hard multiple, can not be obtained by simple shifting and complementation. But rather a full carry propagate addition is required. This algorithm also has the most complex selection logic. Using this algorithm the number of summands is $\lceil \frac{n+1}{3} \rceil$. The extra 1 in the expression comes from the need to ensure that the last summand is a positive multiple of the multiplicand. This is an unsigned operand.



Partial Product Selection Table			
Multiplier Bits	Selection	Multiplier Bits	Selection
0000	+0	1000	-4M
0001	+M	1001	-3M
0010	+M	1010	-3M
0011	+2M	1011	-2M
0100	+2M	1100	-2M
0101	+3M	1101	-M
0110	+3M	1110	-M
0111	+4M	1111	-0

Figure 4: Booth 3 Encoding



Partial Product Selection Table			
Multiplier Bits	Selection	Multiplier Bits	Selection
0000	K+0	1000	K-4M
0001	K+M	1001	K-3M
0010	K+M	1010	K-3M
0011	K+2M	1011	K-2M
0100	K+2M	1100	K-2M
0101	K+3M	1101	K-M
0110	K+3M	1110	K-M
0111	K+4M	1111	K-0

Figure 5: Redundant Booth 3 Encoding

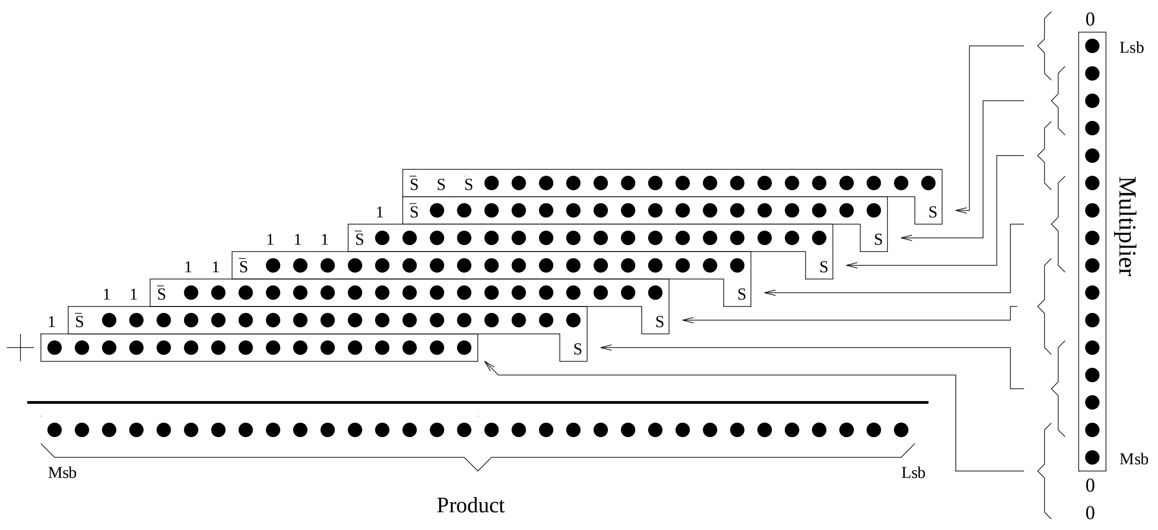


Figure 6: Booth 2 / Booth 3 Encoding

2.4 Redundant Booth 3

Booth 3 has the advantage that it requires a smaller number of summands compared to both non-Booth and Booth 2. Unfortunately, it requires the generation of the hard multiple 3M. Redundant Booth 3 aims to solve this problem by generating an equivalent partially redundant representation [4]. The partially redundant representation is achieved by using a series of smaller adders, with no carry propagation between the adders, as shown in figure 5. A constant is added to each summand so that the positive and negative multiples of the multiplier will have zeroes in the *empty* gaps between the carries. This will allow these locations to be ignored, i.e. there is no need to add the zeroes. The negative of this constant is then added to the summands at design time so that the net result added to the tree is zero.

2.5 Booth 2 / Booth 3

The generation of the 3M multiple causes Booth 3 to be slower than Booth 2. This method reduces the penalty associated with the generation of the 3M by adding some of the summands in parallel, using Booth 2 encoding [5]. Then when the 3M multiple is ready, the remaining summands can be added by using Booth 3 encoding, as shown in figure 6. It should be noted that this method is not a complete win as the penalty for the generation of the 3M multiple can not be completely hidden. This is because of the serialization of the reduction of the summands. That is the reduction of summands using tree structures is logarithmic, and the serialization means the addition of two logarithmic numbers. The number of summands required by this scheme varies between that required by Booth 2 and Booth 3.

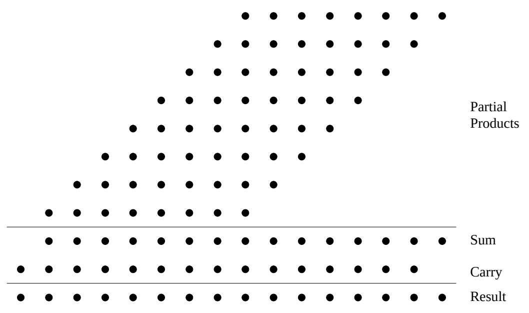


Figure 7: Multiplication Shape.

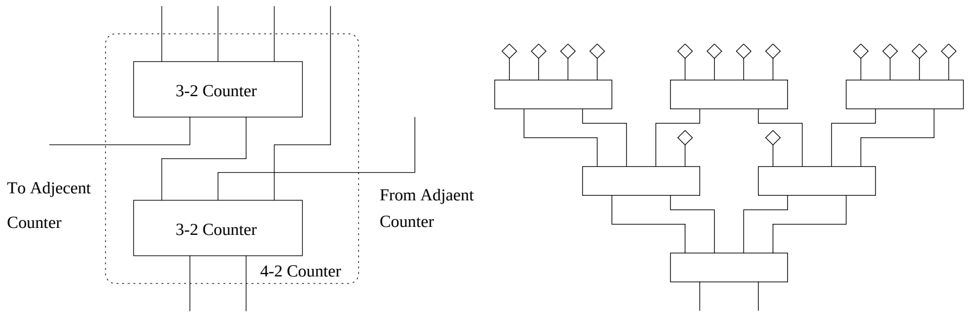


Figure 8: (a) 4-2 Counter Internal structure (b) 4-2 Tree

3 Topology

The different encoding schemes all produce a parallelogram shaped multiplier structure as shown in figure 7. The rows of the parallelogram are added together to produce the final result. In order to decrease the delay needed to produce the result, the redundant form of addition, carry-save addition is used. In this form of addition, carries only propagate to the next column. Each column in this method is treated independently. There are many ways to connect the adder *"counters"* to produce the final result. The number of counters does not differ between the different interconnection schemes. These different methods of interconnecting the counters, or *topologies*, differ in the interconnection scheme used to connect the adders.

These counters can be added together to form a linear array. In a linear array the delay from each of the inputs is proportional to the location at which it is added to the array. however, when all the inputs are available at approximately the same time, there are better solutions. These better solutions are achieved by creating *"balanced delay trees"*. The balanced delay trees are topologies in which the number of stages of delay, or counters, for each input is approximately equal to the number of stages of delay for the other inputs. This is achieved by making the outputs of counters be inputs to non-adjacent counters. These topologies are called tree structures. This is in contrast to linear arrays, in which each counters output is the input for the subsequent counter. These topologies include.

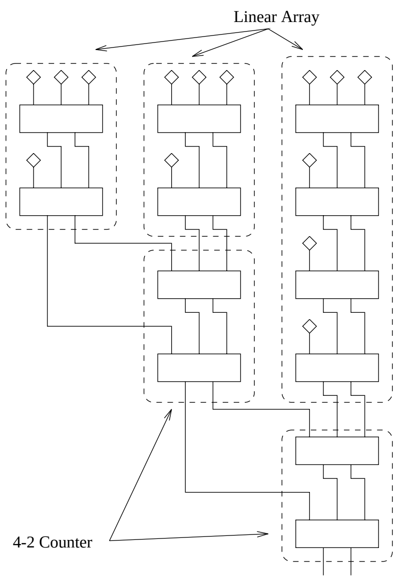


Figure 9: Balanced Delay Tree

3.1 4-2 counter tree

The 4-2 counter is constructed from two 3-2 counters as shown in figure 8a. The 4-2 counter is symmetric, in that it has a 2 : 1 reduction ratio, while the 3-2 counter is not symmetric.

The 4-2 counter tree [6] has a regular and symmetric structure, as shown in figure 8b. In the 4-2 tree, for every four inputs taken at one level, two results are produced at the next level. This can be thought of as a redundant binary tree, since every counter reduces two redundant numbers into one redundant number. The 4-2 tree's binary nature makes it commonly used in pipelined, and iterative multipliers. The symmetric nature of the 4-2 counter facilitates the addition of latches that are needed for pipelining after each 4-2 counter. Iterative and pipelined 4-2 counter trees use the same structure for each bit pitch.

The 4-2 counters do not have a constant requirement for wiring tracks. The number of wiring tracks increases by two when the number of partial products is doubled. Their wiring requirement is similar to that of Wallace trees [8]. In that they are both logarithmic. However, the growth rate of wiring tracks in 4-2 trees is smaller. Also, their wiring requirement is more regular, since Wallace trees, which use 3-2 counters, are extremely irregular making them notoriously difficult to layout.

The advantage of the binary tree reduction of the 4-2 trees is not all that significant for IEEE double precision numbers since the significand size is not a power of two.

3.2 ZM tree

This is the balanced delay tree proposed by Zuras and McAllister [7]. The ZM tree is based upon the idea of balanced delay chains of counters. Trees are constructed by combining progressively longer serial chains into serial chains below them. The connection between

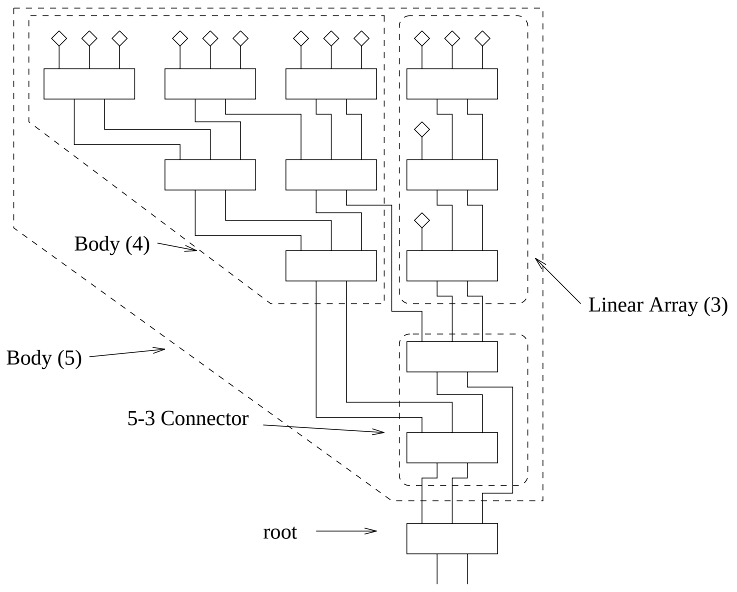


Figure 10: Overturned Staircase Tree

the two chains is made when the total delay of the upper chain is equal to the delay of the lower chain. The connection is made when the number of counters in the critical path of upper chain of counters is as long as the delay of the critical path of the chain of lower counters. This method builds ZM trees of type one, which require only two tracks to feed the output of one counter to the input of a non-adjacent counter, as shown in figure 9.

This tree structure has a very regular layout and it requires only a few primitive cells. This type of tree generally uses more levels of counter delay than the Wallace tree [8] gives, for most values of partial products that must be summed. To reduce the number of levels, higher order ZM trees are constructed, by iteratively replacing the largest chains with ZM trees of type 1. These higher order trees require a larger number of tracks, and are less regular. The number of tracks required by ZM trees is $2P$, where P is the order of the tree, and the number of levels is $O(N^{\frac{1}{P+1}})$.

ZM trees are not easily pipelined. The pipelining of a ZM tree requires that the outputs of the Booth muxes that are not at the first level, ie. those Booth muxes whose output is after the first latch, must be latched in addition to the outputs of the 3-2 counters. So the number of latches required is greater than the number of latches in a 4-2 counter tree. ZM trees can be built to produce the result iteratively using structure that is similar to 4-2 tree.

3.3 OS tree

This is the Overturned Staircase Tree that was proposed by Mou and Jutand [9]. It is called an overturned staircase because the way the counters are connected resembles a staircase. This method divides a tree into a body and a root. The root is the last 3-2 counter in the tree. The body is constructed recursively. In that a body of height k, where k is the

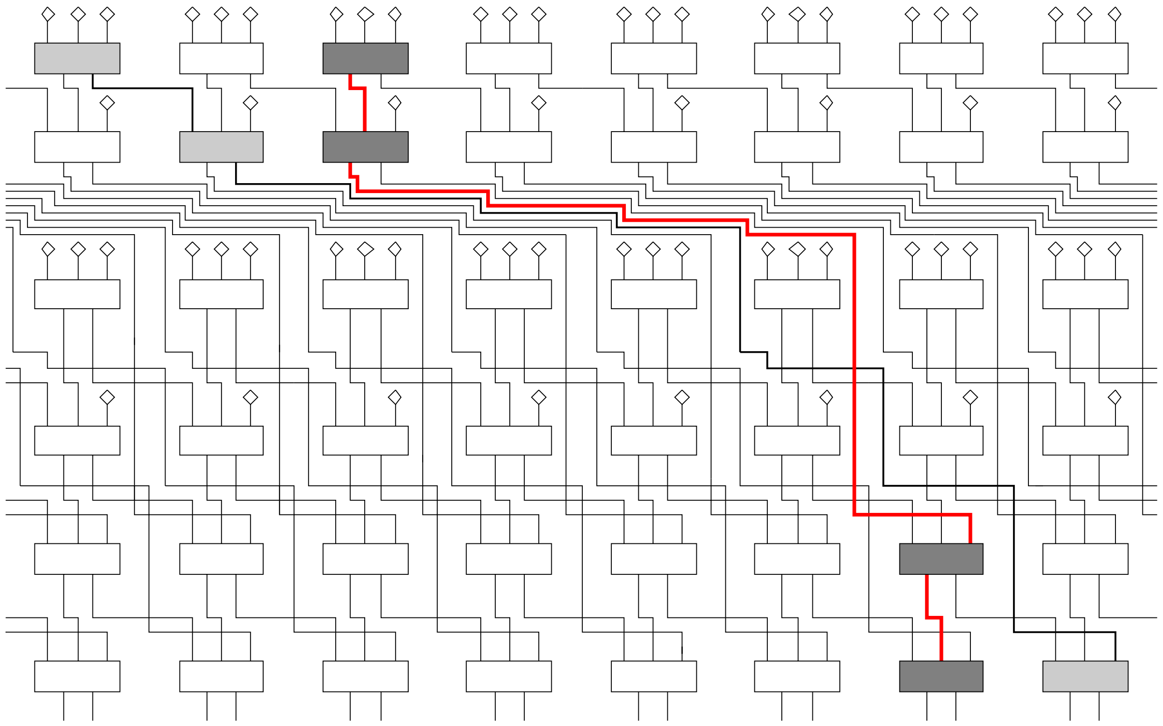


Figure 11: Higher Order Arrays Structure

number of 3-2 counters in the critical path, is constructed from a body of height k-1 and a linear array of height k-2. The linear array and the body are joined using a 5-3 counter. The 5-3 counter is constructed from two 3-2 counters in series. This method build OS trees of type one, as shown in figure 10. This tree structure requires a few primitive cells. It requires 3 tracks to route signals between non-adjacent counters. The OS tree uses more wiring tracks than the ZM tree. The OS tree needs more primitive cells, and it has a less regular structure, compared to the ZM tree.

OS tree structure can give the optimal (minimum) number of counter levels for most numbers of partial products. However, to achieve this, one has to use higher order OS trees. Higher order OS trees can be built by replacing the linear arrays with OS trees of type one. The higher order trees require more wiring tracks. The number of tracks required by OS trees are $3P$, where P is the order of the tree, and the number of levels is $O(N^{\frac{1}{P+1}})$.

OS trees are not easily pipelined. The pipelining of a OS tree requires that the outputs of the Booth muxes that are not at the first level, ie. those Booth muxes whose output is added to the outputs of the first level counters, must be latched in addition to the outputs of the 3-2 counters. So the number of latches required is greater than the number of latches in a 4-2 counter tree. OS trees can be built to produce the result iteratively using structure that is similar to ZM tree. However, OS trees are not typically used for iterative multipliers, since 4-2 trees give a more regular topology, that uses the same number of counter levels.

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

Figure 12: IEEE Double Precision Format

3.4 Higher Order Arrays

This is a class of arrays in which the 3-2 counters are designed as several linear array chains. The chains are combined in parallel when the delay of the upper chain is equal to the delay of the lower chain. This class of arrays can in fact be thought of as a collection of ZM trees of type one. The ZM trees have been designed for the column with the largest number of inputs. This design is replicated for all other columns. In this design the non-critical columns are not optimized. This design trades of the performance of the non-critical columns for regularity, as shown in figure 11. The regularity of the higher order tree is proportional to the number of linear arrays that are combined. The smaller the number of arrays the more regular the design.

Higher order trees can be classified according to the lengths of the chains of partial products before the combining occurs. For example the 6-6-8-8 array has a linear array that combines 6 partial products which is combined with an array the combines 6 partial products. The resulting structure is then combined with an array that combines 8 partial products. Finally the resulting structure is combined with an array that sums 8 partial products.

Higher order arrays are just as easily pipelined as arrays. However since their design is proposed to reduce the latency of the multiplier using the smallest number of wiring tracks available, pipelined iterative higher order trees are not very attractive.

Since Higher order arrays are just ZM trees of type 1 they require only two tracks and there summing time is $O(\sqrt{N})$

4 Layout Issues

The multiplier under consideration uses the IEEE floating point arithmetic standard [2]. The format for double precision numbers, as defined by the standard, is shown in figure 12. 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.

The multiplier is part of a processors datapath which forces the width of each subcell or *bit pitch* to be constant. The required structure for connecting the counters for each topology is achieved by varying the interconnection network of the adders. The interconnection network is routed on top of the adders themselves. For the bit pitch chosen for the study, $45\mu m$, sixteen wiring tracks per bit pitch are available. Only twelve of these tracks are available for routing. The other four tracks being used for the routing of the two operands, result, power, and ground buses. The power and ground can be designed such that they use a single bus by mirroring[1]. These twelve tracks are used to route the interconnections between the counters, in addition to the routing of the Booth muxes outputs and inputs.

Based upon the number of wiring tracks one has available and the number of tracks required by the chosen topology, one uses either single-ended or complementary signal circuits. Single-ended signals include both the static or pass transistor logic families [10]. While the fast complementary signal circuits include the domino [11], NORA [12], and CVSL [13] logic families. An expanded discussion about the merits and disadvantages of each logic family when implementing counters can be found in Song[14].

For the topologies chosen, higher order arrays were implemented in domino logic, while all other topologies used a combination of static and pass transistor circuits.

5 Results

The circuits were simulated using HSPICE. They were simulated for an HP $0.8\mu m$ processes. The simulations are run for typical processing conditions at 25°C. The simulation includes the wire delays that are modeled using the Ersatzco [15] wire model. This model calculates the wire RC delay by placing half the wires capacitance on each side of the wires resistance. The capacitance is calculated using the parallel plate model, with fringing capacitance. This model has the advantage of being computationally simple, while still providing accurate results. The transistor models include an approximation of the gate and source capacitances that is calculated automatically by HSPICE.

The delays are measured from the time the input is latched into the circuit by the system clock in the latches to the time the result becomes available at the output of the trees before the CPA. The specified areas are only for the multipliers reduction tree.

5.1 4-2 Tree

The results for the 4-2 tree are given in figure 13. From this figure we can see that Non-Booth has a larger set up time than Booth 2. This is surprising since it has very simple selection logic. This setup delay is larger because of the extra wiring delay and capacitance due to the large number of summands. The reduction time is also slow because the tree needs 5 “4-2” levels to produce the result. Booth 2 is is the fastest. This is due to several factors. The first is it has a small number of summands, so it does not load the drivers for the booth muxes inputs. Secondly it does not require the generation of a hard multiple. Finally, it requires 8 levels. Booth 3 has a smaller reduction time than Booth 2, even though they have the same number of levels. This is due to its having a smaller number of summands and consequently less capacitance due to wiring. However, the generation of the 3M multiple cause this configuration to be slow. For the redundant Booth 3 configuration the large set up time is due to the complexity of the encoding and Booth muxes. There is a slight advantage to the reduction time compared to Booth 2 due to the decrease in the number of summands. In the hybrid Booth 2 / Booth 3 configuration Booth 2 reduces 18

[1]Mirroring is the circuit layout scheme,in which one places the power and ground buses at the edges of the subcell. The subcells are then mirrored, so that one can place the power and ground lines of adjacent cells on top of one another.

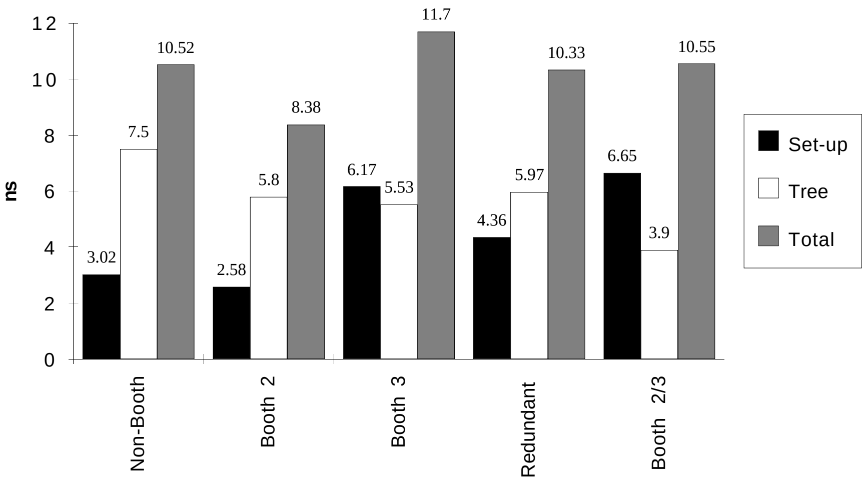


Figure 13: 4-2 Tree Latency

summands. This requires 8 levels, the time required to reduce the 8 summands is greater than the time required to generate the 3M multiple and it is included in the set up time. In this configuration the tree time is the time required to reduce the Booth 3 tree for the configuration.

5.2 ZM Tree

For the balanced delay tree, Non-Booth is the slowest, as can be seen from figure 14. It also has a larger set up time than Booth 2. This is due to the extra wiring delay and capacitance due to the large number of summands. The reduction time is also slow because the tree needs 11 levels to produce the result. Booth 2 is is the fastest. This is due to the same factor as the 4-2 tree Booth 3 has a smaller reduction time than Booth 2. This is due to its having a smaller number of summands and consequently less capacitance due to wiring, and to its needing 7 levels compared to 8 that are needed by Booth 2. However, the generation of the 3M multiple causes this configuration to be slow. For the redundant Booth 3 configuration the large set up time is due to the complexity of the encoding and Booth muxes. There is a slight advantage to the reduction time compared to Booth 2 due to the decrease in the number of summands, which causes the wire capacitance to be less even though they have the same number of levels. In the first hybrid Booth 2 / Booth 3 configuration, Booth 2 reduces 12 summands then it has to wait for the 3M multiple to be generated so that the reduction can continue using Booth 3. This fact causes it to be slightly slower than the other Booth 2 / Booth 3 configuration. The second hybrid Booth 2 / Booth 3 configuration Booth 2 reduces 18 summands. This requires 7 levels. The time required to reduce the 18 summands is still less than the time required to generate the 3M multiple. This design is slightly faster than the other hybrid configuration because it has fewer summands.

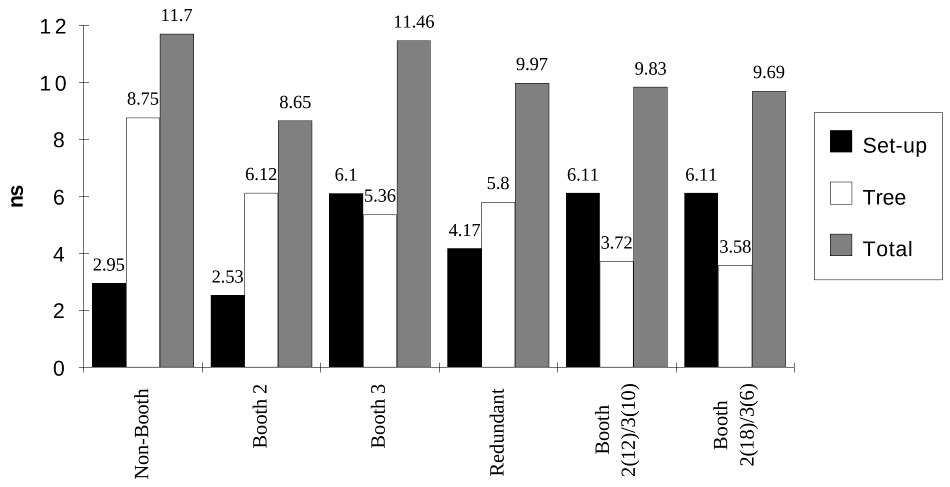


Figure 14: Balanced Delay Tree Latency

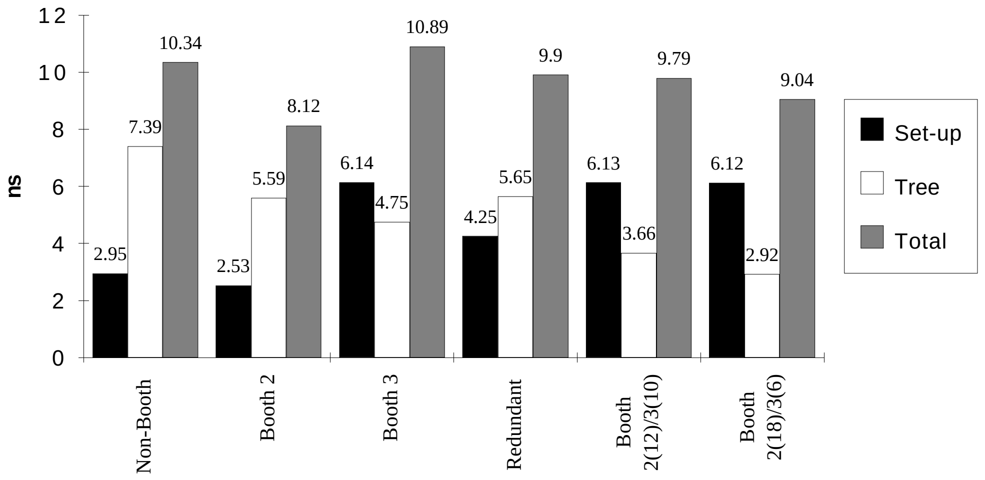


Figure 15: Overturned Staircase Tree Latency

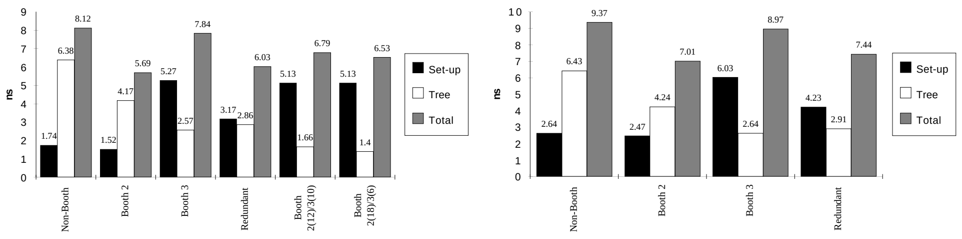


Figure 16: (a) Fully Dynamic HA (b) Dynamic Tree HA

5.3 OS Tree

The same general considerations as the previous two cases apply here as can be from figure 15. However, here Non-Booth is not the slowest. This is because the optimal number of levels is used to produce the result (9 levels). This fact overcomes the extra delay associated with non-Booth and cause Booth 3 to be the slowest.

As an aside, when the wire lengths were zero, that is the extra delay due to the wiring capacitance is removed the difference between non-Booth and Booth 2 in terms of delay becomes 0.03ns. This is an insignificant number that can be ignored. So in fact their delays become equal. That is, for Booth 2; wires contribute only 13.5 % extra delay, while for non-Booth they contribute 32 % extra delay. Therefore the big performance advantage for Booth 2 disappears when there is not any wiring delay. For Booth 3 the wires contributed 12 % of the delay. This contribution is relatively minor, due to the shorter wires used in the tree and to the short wires and many levels of carry lookahead used in the 3M generator, which is the major contributor to the large set-up time. Redundant Booth 3 showed a smaller contribution for the wires to the delay, for they contributed 9.5 % of the delay. Redundant Booth 3's wires contributed to the delay is insignificant because of the short wires used in the ripple carry adders used to produce the redundant 3M representation. In addition the tree reduction time is almost equal for Booth 2 and Redundant Booth 3 when there are no wires. this is because they both use the same number of counter levels. The smaller latency of the tree for the redundant Booth 3 was due to the smaller number of summands and hence, shorter wires. Finally for the combined Booth 2 / Booth 3; the wires contributed 10 % of the delay for both configurations. Their wires contribution to the delay was less than Booth 2 and Booth 3 because of the parallelism between Booth 2 reduction and Booth 3 multiple generation, which hid some of the wire delay.

5.4 Higher Order Arrays

Higher Order arrays are in reality ZM trees of type 1. This means that they are not fully optimized trees and consequently they have a large number of levels. The fact that they are ZM 1 trees means that they have minimal wiring tracks need. This allows the use of dynamic circuits in contrast to the previous methods which all required single ended static circuits. Figure 16(a) gives the latencies for the different algorithms when the multiplier is build from a fully dynamic structure. The same general considerations as the previous three cases apply here. However, the redundant Booth 3 solution is extremely attractive here, as it provides almost the same performance as Booth 2 trees. This is due to the fact that there is a high correlation between the number of summands and the number of levels that are required to reduce them.

For the dynamic tree only part, shown in figure 16(b), it is not possible to intermix Booth 2 and Booth 3 because of the monotonic signal requirement for domino circuits, which would require an extra clock. The addition of an extra clock is not practical due to timing constraints. The same general considerations as those of the fully dynamic multiplier also apply to the multiplier that has only a dynamic tree.

Circuit	Type	Length (μ)	Width (μ)
3-2 Counter	Static	73	45
	Dynamic	105	45
4-2 Counter	Static	146	45
	Dynamic	210	45
AND Gate	Static	16	45
	Dynamic	16	45
Booth 2 Encoder	Static	32	213
	Dynamic	40	369
Booth 3 Encoder	Static	50	284
	Dynamic	60	476
Booth 2 Mux	Static	32	45
	Dynamic	40	45
Booth 3 Mux	Static	50	45
	Dynamic	60	45
56 bit Adder	Static	345	45
	Dynamic	315	45

Table 1: Subcell Circuit Sizes

5.5 Areas

The areas for each subcell used in the design are given in table 1. Non-Booth always has the largest area, as can be seen from figure 17. This is because of the large number of summands required. Booth 3 is always the smallest because it requires the fewest summands. However, the area for Booth 2 is not that much more than redundant Booth 3. This is because of the need for an adder in redundant Booth 3, and the fact that the Booth muxes and encoders are larger for Booth 3. The area for Booth 2 is smaller than that for Booth2(18)/3(6) because the reduction in the number of summands does not offset the extra area used for the 3M adder and the increase in size due to the Booth Muxes. The break even point where the areas are offset occurs at Booth2(12)/3(10).

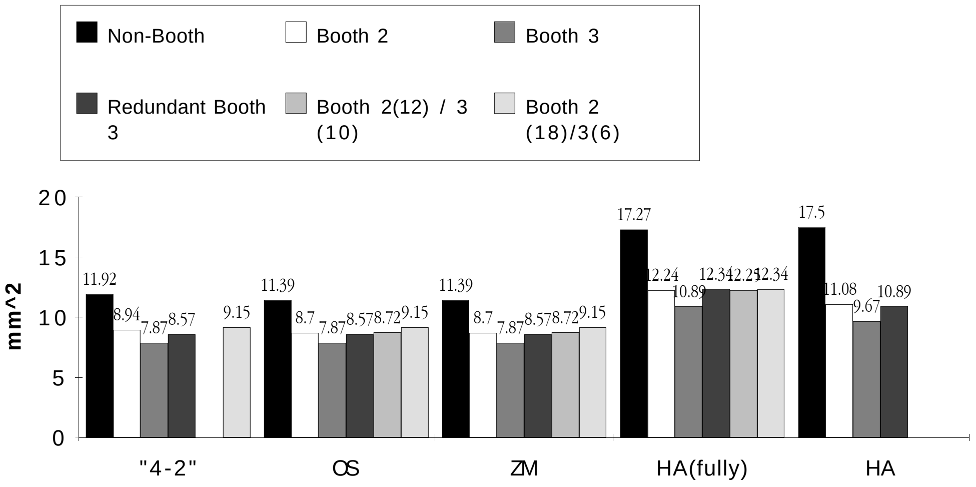


Figure 17: Overturned Staircase Tree Latency

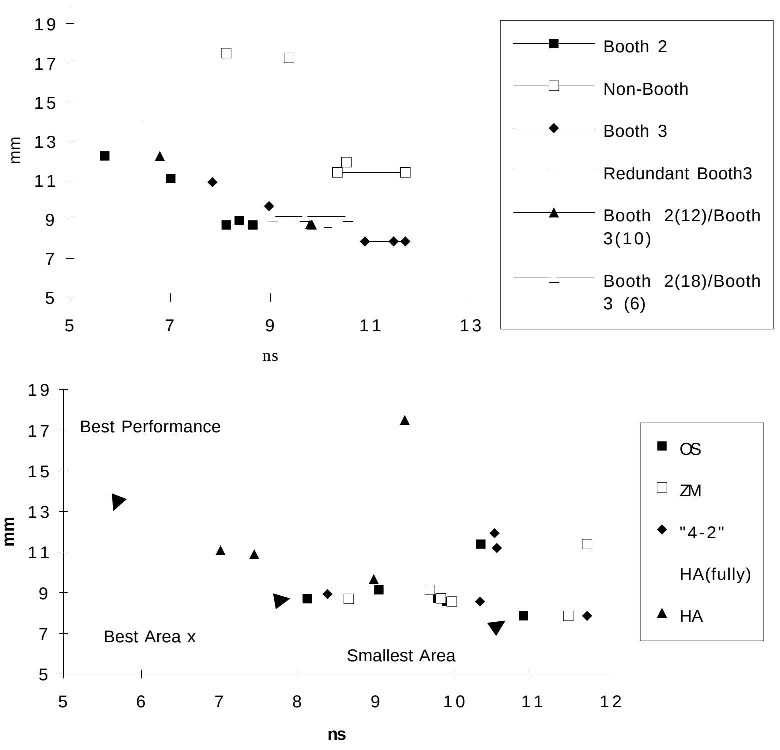


Figure 18: Area vs. Latency: (a) Encoding Scheme (b) Topology

5.6 Area vs. Delay

Figure 18a gives the graphs for the area vs delay for the different algorithms. From this figure we can see that non-Booth is not a viable solution, in that its points are all in the upper right part of the graph. Booth 2 provides the best performance, while Booth 3 has the minimum area. The other 3 algorithms fall in between these two.

Figure 18b gives the graphs for the area vs delay for the different topologies. From this figure the absolute best performance possible is for Booth 2, fully dynamic higher order area. The figure also shows that the smallest area is achieved by Booth 3 overturned staircase tree, while the best performance area is provided by Booth 2 overturned staircase tree.

6 Conclusion

Non-Booth is not a viable design. It consistently gives the largest area, and always is one of the worst in terms of latency. Booth 2 gives the designs with the smallest latency. This is because an additional adder is not required, and that the best possible reduction in number of counter levels needed to sum the summands, as achieved by Booth 3, is only 1. Booth 3 produces the smallest designs because they have the least number of summands. Redundant Booth 3 is not very attractive for tree based designs. It is more suited to standard cell based designs, in which higher order arrays can be thought of as an extreme case, because the number of levels is more closely correlated with the number of summands. Booth 2 / Booth 3 falls in-between Booth 2 and Booth 3 in terms of latency, even though the reduction in area is not as extreme.

As wires continue to account for larger fractions of the total delay, due to decreasing feature size, Booth 3 may provide the best solution. This observation was made because for Booth 2 and Non-Booth the wire contribution to the delay is not insignificant. In the future it is possible that when the wire delay will dominate the total delay, the number of tree levels will not be the deciding factor. Rather, the number of summands, which is directly correlated to the wire lengths, will be the determining element.

References

[1] Stuart Oberman and Michael Flynn, "Design issues in Floating Point Division", *Technical Report: CSL-TR-94-647*, Stanford University Dec 1994.

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

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

[4] G. Bewick, "Binary Multiplication Using Partially Redundant Multiples",*Technical Report: CSL-TR-92-528*, Stanford University, June 1992.

[5] B. S. Cherkauer and E. G. Friedman, "A Hybrid Radix-4/Radix-8 Low Power Signed Multiplier Architecture".

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

[7] 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

[8] C. S. Wallace, "A Suggestion for a Fast Multiplier", *IEEE Trans. Electronic Computers*, pp. 14-17, Feb. 1964.

[9] 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.

[10] C. A.Mead and L. A. Conway, "Introduction to VLSI systems", Reading, MA, Addison Wesley, 1980.

[11] R. Krambeck, C. Lee and H-F. Lew, "High Speed Compact Circuits with CMOS", *IEEE Journal of Solid State*, pp. 614-618, June 1982.

[12] N. Goncalves and H. DeMan, "NORA: A Race Free Dynamic CMOS technique for Pipelined Logic Structures", *IEEE Journal of Solid State Circuits*, Vol SC-18, No.3, pp. 261-266, June 1983.

[13] L. Heller, W. Griffin, J. Davis and N. Thomas, "Cascode Voltage Switch Logic: A differential CMOS Logic Family", *IEEE International Solid State Conference*, pp. 16-19, Feb. 1984.

[14] P. Song, "New circuit and structures for combinatorial multipliers", *Ph.D. Thesis Stanford University*, 1993.

[15] M. Horowitz, EE371 Class notes.