A FAST HYBRID MULTIPLIER COMBINING BOOTH AND WALLACE/DADDA ALGORITHMS

Brian Millar and Philip E. Madrid
Motorola Inc.
6501 William Cannon Drive West
Austin, TX 78741

Earl E. Swartzlander, Jr.
Dept. of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712

Abstract

Using radix 4 recoding, a very fast multiplier is designed which takes advantage of the most desirable characteristics of Booth and Wallace/Dadda multiplier schemes. This hybrid multiplier is shown to be superior from a performance standpoint to the traditional Wallace/Dadda multiplier and consequently, superior to the traditional Booth multiplier as well. Different methods of further increasing the speed are also suggested. Radix 4 is chosen because it is optimal for such a multiplier. This will be explained by showing the limitation of increasing the radix to radix 8.

1 Introduction

There are many methods to perform binary multiplication of two numbers. Some of the algorithms strive for minimization of gate count, layout area, or interconnect. Others are designed to be extremely fast, but increase the size of the multiplier significantly. Two well known multiplication algorithms, Booth higher radix multiplication [1,2,3], and Wallace/Dadda multipliers [4&5], when combined in a hybrid fashion, can yield a multiplier that is faster and not more than 30% larger than the traditional Wallace/Dadda multiplier. A similar idea has been proposed previously [6], but this paper shows the implementation. The higher radix Booth algorithm has the advantage of reducing the number of partial products which are added [2], while the Wallace/Dadda approach allows the partial products to be added very quickly. A hybrid multiplier combining both of these strengths is presented. It is shown to be very fast, and realized with a reasonable amount of logic gates. Section 2 develops a 32x32 bit hybrid multiplier and analyzes the size and gate delay compared with a straightforward Wallace/Dadda multiplier. Section 3 briefly discusses why radices higher than 4 are not advantageous to this multiplier.

2 32x32 Booth Radix 4 and Wallace/Dadda Hybrid Multiplier

A fast 32x32 bit multiplier is designed combining the Booth radix 4 and Wallace/Dadda techniques. The Booth approach generates less partial products than the Wallace/Dadda approach. The Wallace/Dadda method, however, adds the partial products significantly faster than a traditional Booth multiplier which accumulates a running sum of the partial products. It might seem therefore that the higher the radix of Booth implemented, the fewer partial products added by the Wallace/Dadda structure and hence the faster the multiplier. This is not the case, however, as will be understood by the end of this paper.

This hybrid multiplier is readily divided into four major blocks for discussion: a partial product generation block, a partial product selection block, a reduction tree, and a high speed adder.

2.1 Partial Product Generation

The two non-zero numbers being multiplied are A and B, where A is the multiplier word in the form:

$$A=a_S.a_{31}\ a_{30}\ a_{29}\ \ldots\ a_0.$$

and B is the multiplicand in the form:

$$B=b_S.b_{31}\ b_{30}\ b_{29}\ \ldots\ b_0.$$

The numbers have 1 sign bit and 32 binary digits. Table 1 shows the required steps in the Booth algorithm to generate a partial product [2,3]. The appropriate bits of the multiplier word are analyzed at each step, and the described action if any is performed to the multiplicand before it is added to the partial product and right shifted by two bit positions.

Table 1: Booth Radix 4 Partial Product Rules.

Bits a_{i+1},a_i,a_{i-1}	Booth step before shifting
000	Nothing
001	+B
010	+B
011	+2B
100	-2B
101	-B
110	-B
111	Nothing

In hardware, the partial products B and 2B can be generated immediately. B is an input to the multiplier, and 2B is formed by shifting the bits of B to the left one place. -2B can also be generated immediately once -B is computed. On the surface, it appears that -B will require a high-speed adder to compute, and therefore require a large overhead to generate. However, a trick will be used here: only $\overline{B}$ will be formed (the 1's complement). This is easily done with an inverter for each bit. Then, the carry in of one is added in the reduction structure, to complete the 2's complement. A special mux for the LSB enables this carry in to be set when a "-B" partial product is needed.

The important realization here is that adding the carry in to the reduction section does not in any way slow down the reduction, but it does greatly reduce the time needed to generate the partial products. Were it not for this, a high-speed adder would be necessary to generate -B from B. This will be clear from the Dadda reduction structure shown later.

The partial product generation is shown by Figure 1, where a colon is used to denote concatenation of bits. The partial products are extended to 33 bits. For B and $\overline{B}$, the sign bit is

0-7803-0510-8/92$03.00 ©1992IEEE

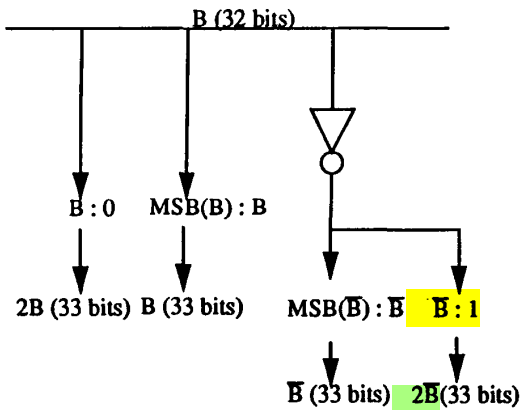


Figure 1. Partial Product Generation.

extended. For 2B, B is shifted left by one unit and a '0' is wired in for a positive operation. For $2\overline{B}$, $\overline{B}$ is shifted left one unit, and a '1' is wired in for a negative operation. Some examples will demonstrate this later.

The partial products are available in 1 gate delay for radix 4. All logic gates, whether they are inverters or NAND gates, will be assumed to have 1 gate delay, indicated as "1Δ." This 1Δ, needed to form all possible partial products, is absorbed in the next block, the partial product selection.

2.2 Partial Product Selection

With the implementation of radix 4 for a 32x32 bit multiplication, there are 17 partial products for the reduction structure (see Figure 2) . Also, there are 33 bits per partial product: B and $\overline{B}$ are sign extended, 2B shifts in a '0' as the LSB, and $2\overline{B}$ shifts in a '1' as the LSB. The LSB mux contains several additional gates to control the carry in signal. The regular muxes and the LSB mux are depicted in Figure 3. As can be seen, the regular muxes and the LSB muxes contain 10 and 13 gates respectively. A fan-in of 6 is not be very desirable here, but with appropriate sizing at the transistor level, this OR gate is not be a problem. The muxes select the appropriate partial product (or 0 for just shift which results if none of the other partial products are selected) based on the bits of the multiplicand being analyzed according to the Booth radix 4 rules already presented.

The general mux structure is shown by Figure 2, where each successive partial product is shifted left by 2 units in the reduction structure. The last partial product requires only 32 muxes instead of 33. This is because the sign bit of the multiplier word is extended (see Figure 2) and according to Table 1, only B or $\overline{B}$ can result for the last partial product. The sign extension to 33 bits is not needed here.

With the muxes as shown, the total delay from the partial products is 3Δ. This absorbs the 1Δ delay for the inverters needed to form $\overline{B}$ and $2\overline{B}$ in 2.1. This is because $\overline{B}$ and $2\overline{B}$ can be formed in parallel with the inverters forming the compliments of the mux select lines. The total gate count for the muxes needed to perform a 32x32 bit multiplication in radix 4

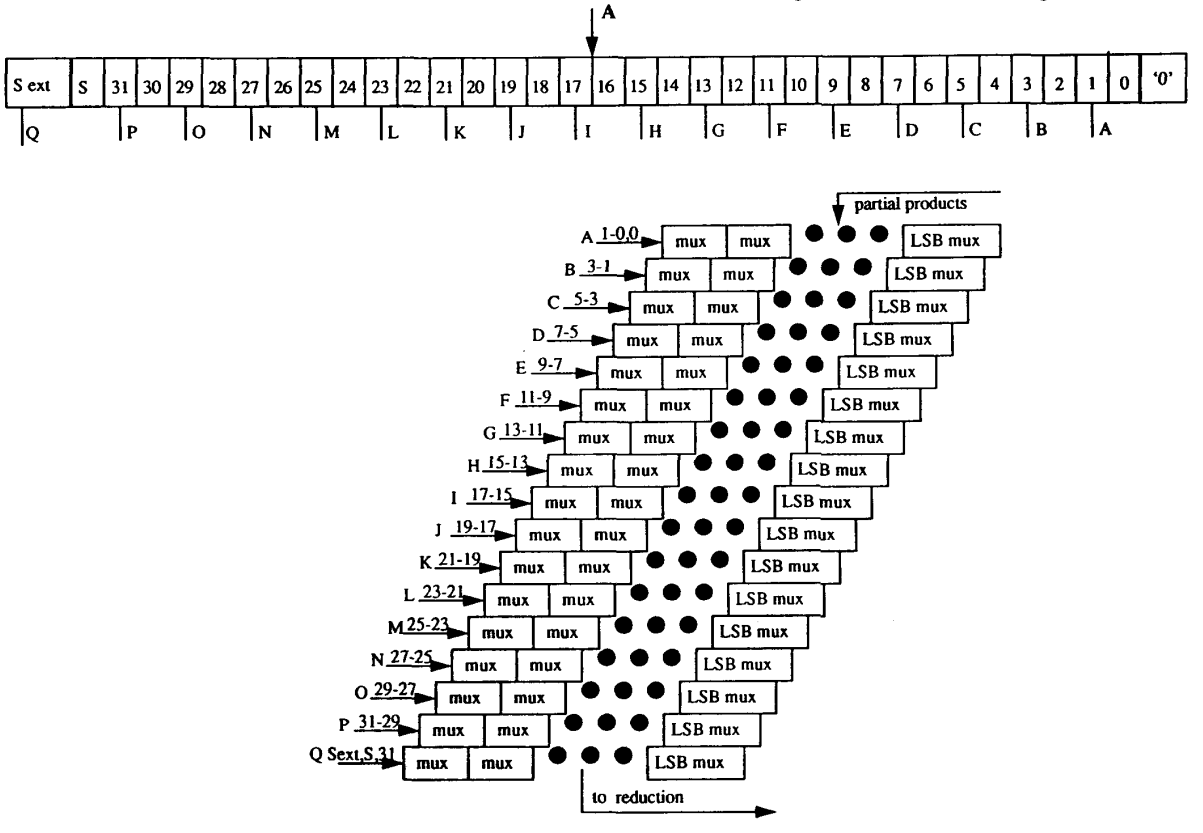
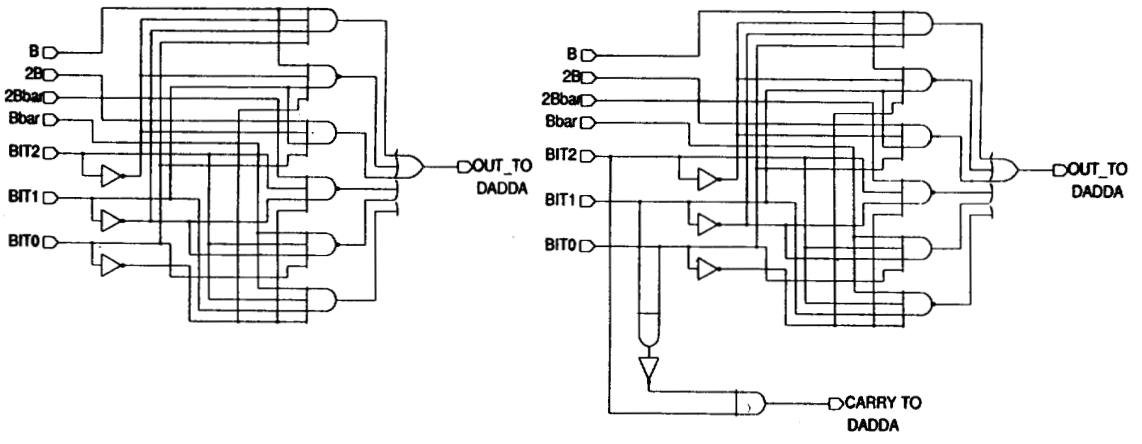


Figure 2. Mux Tree Structure.
For each group of three bits of the multiplier word, the appropriate partial product is muxed according to Booth Radix 4 rules. This builds a mux tree, where each partial product is shifted by 2 bit positions before the reduction structure.



Regular Mux

LSB Mux

Figure 3. Regular and LSB Mux shown in Mux Tree Structure

where PP denotes Partial Product is:

16 PPs	*32 regular muxes*	10 gates /mux =5120 gates
1 PP	*31 regular muxes*	10 gates / mux = 31 gates
17 PPs	*1 LSB mux	* 13 gates / mux = 221 gates
	Total =	5372 gates

2.3 Reduction Stage Followed by a High-Speed Adder

After a delay of only 3Δ, the partial products are available to the reduction structure. There are at most 18 bits per column, including the carry in's. There are only 17 partial products, but one column in the reduction structure has 18 "dots" due to the potential carry-in. Using Dadda's sequence [5] with 18 dots as the limiting factor, 6 reduction steps are required to compress the partial products down to 2 for the high-speed adder. The 6 steps compress the structure down in the following manner:

18->13->9->6->4->3->2.

With a conventional 32x32 bit Wallace/Dadda multiplier, the partial products are available in 1Δ, but there are 32 partial products which requires 8 reduction steps:

32->28->19->13->9->6->4->3->2.

The reduction structure following the mux section is shown in Figure 4. From the figure, several observations can be made. First, there are 17 rows, one for each partial product. Each partial product is sign-extended except for the last one which needs only 32 bits to form a complete 64 bit result. The carry muxes select whether or not $\overline{B}$ or $2\overline{B}$ was formed and needs the carry in to form the true 2's complement. The final sign bit is the xor of the sign bits of A&B. This reduction structure then gets reduced as shown in Figure 5, to a 2 row 64 bit result. Finally, the reduction structure is fed into a 64 bit high-speed adder which produces the final result. In Figure 5, H's and F's are used to represent half and full adders respectively. Typically Dadda "dots" are used to show the reduction steps.

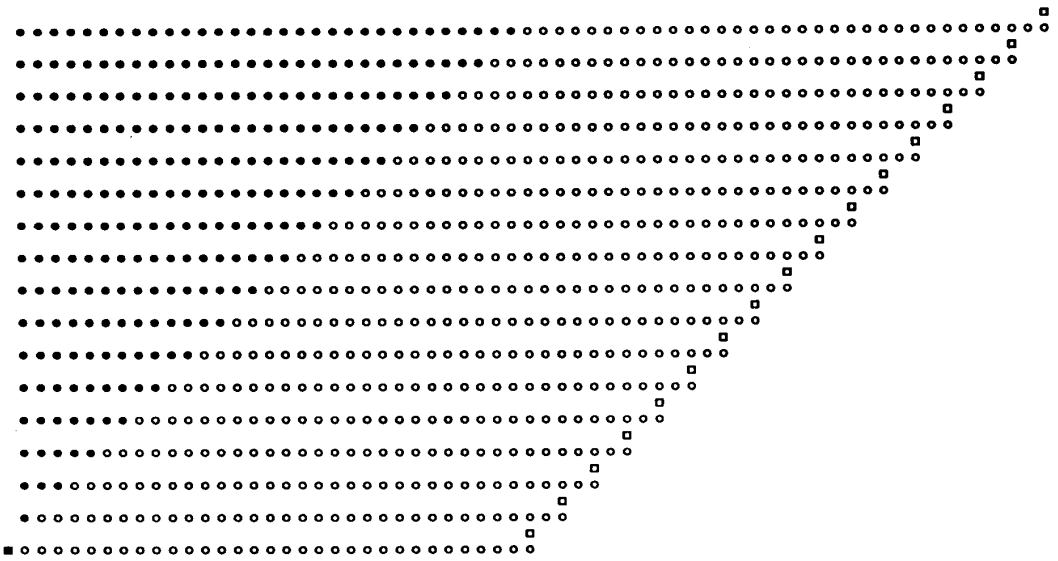


Figure 4. Booth Generated Partial Products.

o = Mux Outputs ◘ = Carry from the mux to complete two's complement • = Sign extended bit ■ = Sign Bit [A(s)⊕B(s)]

Figure 5. Dadda "Dot" Diagram.

F = Full Adder H = Half Adder

2.4 Examples

Several examples follow to illustrate different cases in the multiplier. In the interest of brevity, the examples are smaller than a 32x32 bit multiplication, but still serve to illustrate how the multiplier works. The examples show how the partial products would appear to the reduction block, and the final result.

Example 1

A=11/32=	0.010110
B= 5/16 =	0.010100
$\overline{B}$=	1.101011
Answer = 55/512 =	0.000110111000

A=000101100
(append 0 to LSB in Booth and sign extend)

Partial Product	A bits	Action
1		
111111010111	100	$\overline{2B}$+1
0000101000	011	2B
00010100	010	B
000000	000	None

0.000110111000

0 is XOR of sign bits of A & B

Example 2

A=11/32=	0.010110
B= -5/16 =	1.101100
$\overline{B}$=	0.010011
Answer=-55/512 =	1.111001001000

A=000101100
(append 0 to LSB in Booth and sign extend)

Partial Product	A bits	Action
1		
000000100111	100	$\overline{2B}$+1
1111011000	011	2B
11101100	010	B
000000	000	None

1.111001001000

1 is XOR of sign bits of A & B

Example 3

A=-25/32=	1.001110
B= 5/16 =	0.010100
$\overline{B}$=	1.101011
Answer=-125/512=	1.110000011000

A=110011100
(append 0 to LSB in Booth and sign extend)

Partial Product	A bits	Action
1		
111111010111	100	$\overline{2B}$+1
0000000000	111	None
00010100	001	B
1		
101011	110	$\overline{B}$+1

1.110000011000

1 is XOR of sign bits of A & B

Example 4

A=-25/32=	1.001110
B= -5/16 =	1.101100
$\overline{B}$=	0.010011
Answer=125/512=	0.001111101000

A=110011100
(append 0 to LSB in Booth and sign extend)

Partial Product	A bits	Action
1		
000000100111	100	$\overline{2B}$+1
0000000000	111	None
11101100	001	B
1		
010011	110	$\overline{B}$+1

0.001111101000

0 is XOR of sign bits of A & B

Analyzing Example 1, there are four columns of partial products. The one '1' all by itself is the carry in which is needed to form the 2's complement of $2\overline{B}$. The partial products are sign extended to form the correct number of bits. Notice the last partial product is not sign extended (it is only six bits not seven) since the number of decimal bits needed in the result are already met. Also, the carry out of the most significant decimal digit is discarded. The sign bit is independently calculated as the XOR of A&B.

Example 2 shows the case when B is the negative of Example 1. The same characteristics can be seen here.

Examples 3 & 4 serve to illustrate a couple of points. They both contain a $2\overline{B}$+1 step in them (where $2\overline{B}$+1 is equivalent to 2* 2's complement of B). In Example 3, B is a positive value and in Example 4, B is a negative value. But, as can be seen, when the $2\overline{B}$+1 operation is needed, a '1' is shifted in regardless of the sign of B. Likewise, if a 2B operation were needed, a '0'would be shifted in regardless of the sign. Notice again that the last partial product is not sign extended (it is only six bits not seven) since the number of decimal bits needed in the result are already met. Again the carry out of the MSB is discarded, and the sign is calculated from the xor of A & B.

2.5 Analysis of Delays and Extension to Larger Multipliers

In the following analysis, a standard 9 gate full adder is used which requires 6Δ for the sum to be computed once its operands are present. Also, a standard 4 bit block size/ 3-level carry look ahead adder [7] is used for the 64 bit fast adder.

Using this radix 4 approach, the total delay to compute the product is:

3Δ for partial product generation/selection (muxes)
36Δ for reduction (18->13->9->6->4->3->2)
6 reduction steps * 6 gate delays for full adder
14Δ for final high-speed carry look-ahead adder

53Δ total

This compares to a delay of 63Δ for a traditional Wallace/ Dadda approach:

1Δ for partial product generation
48Δ for reduction
(32->28->19->13->9->6->4->3->2)
8 reduction steps * 6 gate delays for full adder
14Δ for final 64 bit high-speed carry look-ahead adder

63Δ total

The total gate count for this hybrid approach is:

32 for inverters to form $\overline{B}$
5372 for muxes (Figure 2)
6210 for 690 full adders (Figure 5)
248 for 62 half adders (Figure 5)
4 for sign-bit xor made from AND,OR,INV

11866 total gates + a 64 bit high speed adder

This compares to a gate count for a traditional Wallace/ Dadda of:

1024 AND gates for partial products (32^2)
8091 for 899 full adders
124 for 31 half adders
4 for sign-bit xor made from AND,OR,INV

9243 total gates + a 64 bit high speed adder

Thus a 28% increase in complexity results in a 16% decrease in delay for a 32x32 bit multiplier. The percentage increase in complexity would actually be lower if the high-speed carry look-ahead adder was figured into the calculation.

Consider now 64 bits. Using Booth radix 4 there are 33 partial products. For the hybrid approach, the delay is:

3Δ for muxes and partial product generation
48Δ for reduction
(33->28->19->13->9->6->4->3->2)
18Δ for 128 bit high-speed carry look-ahead adder

69Δ total

The regular Wallace/Dadda approach incurs a delay of:

1Δ for partial product generation
60Δ for reduction
(64->63->42->28->19->13->9->6->4->3->2)
18Δ for 128 bit high-speed carry look-ahead adder

79Δ total

This again is a savings of 10Δ. The hybrid multiplier saves reduction steps at the expense of increased area for the multiplexers. The number of steps reduced depends where on Dadda's sequence the number of partial products for both methods fall. For 32 and 64 bits, Booth reduces the number of partial products rows by 2, thereby saving 2 reduction steps. Any savings in reduction steps is very beneficial since each step requires 6Δ with this full-adder implementation.

2.6 Possible Refinements

1) To speed up both the Wallace/Dadda multiplier and the radix 4 hybrid presented, an analysis of a full-adder circuit at the transistor level can be made. With proper sizing, a straightforward "and-or" approach to the sum and carry [6] could yield smaller delay times than the modular fast adder from half adders. This could be a great savings since 6 gate delays could be cut to three for each reduction step. This is the most time consuming part of the algorithm.

2) Another possibility is to consider a faster high-speed adder such as that of a carry select adder.

3) Finally, the gate count could be significantly decreased by using dynamic logic in the multiplexer blocks.

3 32x32 Booth Radix 8 and Wallace/Dadda Hybrid Multiplier

On the surface, it might appear that using a higher radix Booth algorithm would make an even faster multiplier. Higher radices would produce fewer partial products, which in turn would require fewer reduction steps to compress to a two partial product result for the high-speed adder. However, the speed to generate, plus the size to select the partial products is very limiting, as will be seen.

3.1 Partial product generation

Table 2 shows the required steps in the Booth algorithm for radix 8 to generate a partial product. The bits of the multiplier word are analyzed at each step, and the appropriate action if any is performed to the multiplicand before adding to the current partial product and shifting by three bit positions to the left.

From the rules, it can be seen that the following partial products are needed: B, 2B, 3B, 4B, -4B, -3B, -2B, -B. The limiting factors in terms of speed are the partial products: 3B & -3B which cannot be formed by simple shifting of B or -B. To perform this addition, a high speed adder must be used. For a 32x32 multiplication, it takes 14Δ to generate a 33 bit 3B, and an additional 1Δ for the inversion needed to generate 3B.This incurs an accumulative delay of 15Δ just to create the partial products. This turns out to be a rather large overhead.

3.2 Partial Product Selection

For a radix 8 Booth multiplier, 4 bits of the multiplier word, A, are being examined. Based on these four bits, a selection of the appropriate partial product must be made for that step according to Table 2. This requires many multiplexers of the type shown in Figure 6. The mux must select which of B, 2B, 3B, 4B, -4B, -3B, -2B, -B (or 0 for do nothing just shift, but this will be the result if none of the other partial products are selected) should be the partial product for that

Table 2: Booth Radix 8 Partial Product Rules.

Bits $a_{i+2},a_{i+1},a_i,a_{i-1}$	Booth step before shifting
0000	Nothing
0001	+B
0010	+B
0011	+2B
0100	+2B
0101	+3B
0110	+3B
0111	+4B
1000	-4B
1001	-3B
1010	-3B
1011	-2B
1100	-2B
1101	-B
1110	-B
1111	Nothing

group of four bits. A mux is needed for each bit of the 32 bit partial product. In the radix 4 example, the mux had to select between 1 of 6 inputs (B and $\overline{B}$ were repeated). In radix 8, the mux must select between 1 of 14 inputs due to B, 2B, 3B, -B, -2B, and -3B being repeated. This greatly increases the gate count for the muxes, and would increase the delay since a 14 input OR gate would not be very feasible.

3.3 Reduction Feeding a High-Speed Adder

The radix 8 approach to a 32x32 bit multiply generates 11 partial products for the reduction structure. This is obtained by extending figure 2. This is a slight improvement over the 17 generated from radix 4, and a large improvement over the 32 partial products generated by the conventional Wallace/Dadda multiplier.

3.4 Analysis of Delays and Extension to Larger Multipliers

Using the described approach from partial product generation to the end result the total delay for a radix 8 32x32 bit multiplication is:

15Δ for partial product generation
4Δ for partial product selection
30Δ for reduction (11->9->6->4->3->2)
14Δ for a 64 bit high-speed adder

—

63Δ total

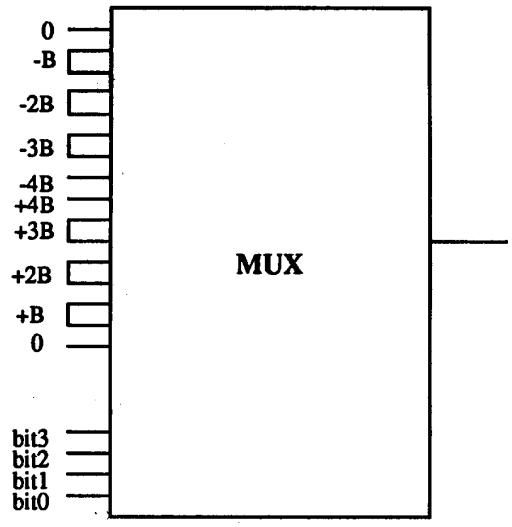


Figure 6. Mux Needed for Booth Radix 8.

From section 2.5, the delay of a straightforward Wallace/Dadda multiplier and of a radix 4 Booth and Wallace/Dadda hybrid is 63Δ and 53Δ respectively. Thus, the savings in Dadda reduction steps is not worth the additional time needed to compute the partial products for radix 8.

Consider 64 bits. The radix 8 would have 21 partial products and a delay of:

15Δ for partial product generation
4Δ for partial product selection
42Δ for reduction
(21->19->13->9->6->4->3->2)
18Δ for a 128 bit high-speed adder

—

79Δ total

Again, from section 2.5, it was shown that a regular Wallace/Dadda multiplier would have a delay of 79Δ, and a hybrid multiplier would have a delay of 69Δ.

Thus, any radix of 8 or higher is not feasible, and a radix 4 approach is the best solution for the proposed hybrid multiplier.

4.0 Conclusion

There will always be a need to perform basic arithmetic tasks such as addition and multiplication as quickly as possible to push the technology barrier further. This paper suggests an approach to improving the speed of a fixed-point multiplier. In section 2, a radix 4 hybrid multiplier was developed combining the strengths of the Booth and Wallace/Dadda methods for multiplication. It was shown that for a small sacrifice in gate count, a multiplier could be developed that was faster than a conventional Wallace/Dadda multiplier. Section 3 shows how the multiplier would be developed in radix 8. It shows that the significant increase in gate count and the increase in delay required for partial product generation make radix 4 best suited for this hybrid multiplier. Finally, additional ideas to increase the speed of the multiplier were suggested for future study.

References

[1] A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, pp. 236-240, 1951. Reprinted in [8] pp. 100-104.

[2] H. Sam and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Applications in Multiplier Implementations," IEEE Transactions on Computers, vol. 39, pp. 1006-1015, 1990.

[3] O. L. MacSorley, "High Speed Arithmetic in Binary Computers," Proc. IRE, vol. 49, pp. 67-91, 1961. Reprinted in [8] pp. 14-38.

[4] C. S. Wallace,, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, vol. EC-13, pp. 14-17, 1964. Reprinted in [8] pp. 114-117.

[5] L. Dadda, "Some Schemes for Parallel Multipliers," Alta Freq., vol. 34, pp. 349-356, 1965. Reprinted in [8] pp. 118-125.

[6] S. Vassiliadis, E. M. Schwartz, and D. J. Hanrahan, " A General Proof for Overlapped Multiple-Bit Scanning Multiplication," IEEE Transactions on Computers, vol. 38, pp. 172-183, 1989.

[7] A. Weinberger and J. L. Smith, "A logic for High-Speed Addition," National Bureau of Standards Circular 591, 1958, pp. 3-12. Reprinted in [8] pp. 47-56.

[8] E. E.Swartzlander Jr., ed., Computer Arithmetic, vol. 1, Los Alamitos, Ca: IEEE Computer Society Press, 1990.