

A New Multiplier Using Wallace Structure and Carry Select Adder with Pipelining

Reza Hashemian, Senior Member IEEE
 Northern Illinois University
 Dept of Electrical Engineering
Reza@ceet.niu.edu

Abstract- Design of a high performance and high-density multiplier is presented. This multiplier is constructed by using the Wallace tree structure with pipelining. A fast carry select adder is used for the final two-operand adder. It is shown that the time delay for the entire multiplier is $O(\log(n))$. The design is particularly carried out for a 32-bit multiplier with two sections of pipelining, to balance the delays through the multiplier. A total of ten gate delays is all needed for the multiplication throughput.

Index Terms – Multiplier, Wallace structure, Carry-select adder, Pipelining.

I - INTRODUCTION

Multipliers are among the most frequently used devices in digital signal processing. However, the delays associated with high-density multipliers are typically large and this is the main cause of slow processes in the data manipulation [8-12]. Our objective here is to use the Wallace tree structure [1-5] with pipelining, and design high-density multipliers that have high-speed throughput with delays not linear but logarithmically proportional to the data size. The second objective is to minimize the array of (3, 2) adders in each layer of Wallace structure. Yet, the third objective is to maximally balance between the time delays associated with each section of the pipeline. It is shown that, through a two sections pipelining a balanced delay between sections is best achieved. In fact, in a 32-bit multiplier that uses Wallace structure [3, 4], followed by a carry select two-operand adder [6] the total delay in each section, and hence that of the multiplier's throughput, is reduced to ten (two-input NAND) gate-delays.

II – DESIGN DEVELOPMENT

Although the proposed algorithm is valid for multipliers of any size, but with this scheme higher performance (high speed and reduced hardware) is better achieved for higher density multipliers. In particular, for a given two 32-bit operands $A = a_{31} a_{30} \dots a_1 a_0$, and $B = b_{31} b_{30} \dots b_1 b_0$ the multiplication product $C = A*B$ is obtained by going through three stages of operations. The first stage is going through an AND-array generating single bit multiplications; the second stage consists of a full adders tree, known as Wallace structure [3], and the third stage consists of a high speed two-operand adder generating the final product C [6]. The *AND-array*, is constructed from a two dimensional 32 by 32 array of 2-input AND gates producing $a_i b_j$, for i and $j = 0, 1, \dots, 31$. All $a_i b_j$ terms are produced in one gate-delay. The next stage in the operation is the Wallace structure.

Wallace structure

Wallace structure [3] consists of multi-layers of mixed adders (mixed full and half adders), MA. The adders in each layer operate simultaneously, but those in different layers operate in sequence. Now, if $n(h)$ and $n(h-1)$ represent the number of entries (bits) in the corresponding columns in layers h and $h-1$ we can write [2]

$$n(h) = \lfloor 3h(h-1)/2 \rfloor \quad (1)$$

If we ignore the floor operator, we can conclude that the number of entries in the corresponding columns is reduced by a factor of 1.5, from one layer to the next. This reduction continues from the top layer to the bottom, until the last layer that contains only one or two bits per column. Figure 1(a), symbolically, shows the Wallace structure for a 4 by 4 multiplier.

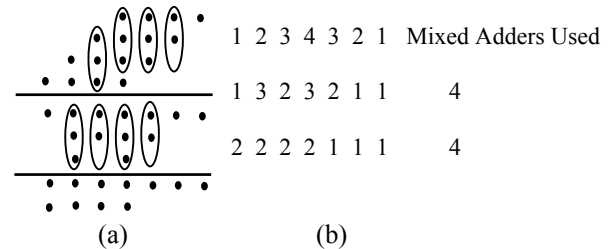


Fig.1 – (a) The Wallace structure for a 4-bit multiplier, and (b) The column entries and the number of MAs in each layer.

Next, we can calculate the signal delay through the Wallace structure. Referring to Fig.1 (a), if we consider the time delay of a full adder equivalent to two gate-delays, then for a two-layer Wallace structure the total delay amounts to four gate-delays. Figure 1(b), shows the listing of the number of bits in each column of each layer in the Wallace structure. In addition, the total number of mixed adders (MA) used in each layer is shown.

Figure 2 is very similar to Fig. 1(b), except it is generated for a 16-bit multiplier rather than a 4-bit. As shown, there are 6 layers of full adders used in this structure and the total delay here amounts to 12 gate-delays. Note that, when one layer in the structure performs the (3, 2) reduction the other five layers have to stay idle and wait for their turn. Pipelining helps to solve this problem, where, we can simultaneously perform the reductions in different layers and store them, and as a result, speed up the process of getting the throughput. However, there are some disadvantages in using pipelining, and there are some costs involved. For each pipelining, we need to have register arrays equal to the size of the data being available. In addition, to make the data flow cyclic we need to use the clock and wait for stable data to get stored, which adds to the time delay.

Finally, the register delay is also added to the overall delay in the data flow. As a result, for an optimal pipelining we need to reduce the number of pipeline sections to a minimum where the saving in time is worth the cost of the pipelining. Even for 16-bit multiplier our experience indicates that the pipelining may not be very cost effective. This is mainly because the delay through the Wallace structure does not match the delay through the final stage, i.e., the two-operand adder.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	3	2	3	4	5	7	6	7	9	8	9	11	10	11	9	10	9	7	8	7	5	6	5	3	4	3	2	1	1	MA=76
2	1	3	2	4	3	5	4	6	5	7	6	8	7	8	6	7	5	6	4	5	3	4	2	3	2	1	1	1	MA=51	
2	2	1	3	3	2	4	4	3	5	4	6	5	6	4	4	5	3	3	4	2	2	3	2	1	1	1	1	MA=33		
2	2	2	2	1	3	3	3	3	2	4	4	4	4	3	3	3	3	4	2	2	2	2	3	2	1	1	1	1	MA=24	
2	2	2	2	2	2	2	1	3	3	3	3	3	2	2	2	2	2	2	2	2	2	3	2	1	1	1	1	1	MA=16	
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	MA=16	

Fig.2 –The Wallace structure for a 16-bit multiplier, and the number of MAs in each layers.

32-bit multiplier with pipelining

For a 32-bit multiplier the situation is somewhat different from that of a 16-bit multiplier. Here, we have 8 layers of full adders used in the Wallace structure. This essentially means that, if we choose to go directly from the AND-array to the final two-operand adder we need to get through at least 16 gate-delays. On the other hand, a fast 64-bit carry select adder [6] takes only 8 gate-delays to add, and thus, for each multiplication the two-operand adder is sitting idle for two third of the time. This suggests for an effective pipelining to speed up the throughput. In fact, with two sections of pipelining we are able to reach to an optimal performance for the multiplier and a cost-effective design. Now, we need to decide on the best locations to put the registers in the pipeline.

A simple calculation reveals that close to 25 gate-delays are needed to complete a multiplication of 32-bit operands without pipelining. From this number 16 gate delays belong to the Wallace structure, one gate delay to the AND-array and 8 gate delays belong to the 64-bit carry select adder. Now, with addition of three registers, each carrying one-gate delays, we come up with overall 28 gate delays. Hence, if we split this delay into three, then each stage of the pipeline takes about 10 gate delays to complete.

The next step is to locate the positions of the registers within the data path. The best location for the first register, R1, is after the fourth layer in the Wallace structure, as shown in Fig.3. This is because, with one gate delay for the AND-array, eight gate delays for the first four Wallace layers, and one gate delay for the register R1 we get exactly to the limit of 10 gate delays. Similarly, the best location for the second register, R2, is at the end (8th layer) of the Wallace tree, where the data is prepared for the two-operand adder. This stage takes 9 gate delays, i.e., 8 for the last four layers in the Wallace structure, and one for the register R2. Finally, the third register, R3, is placed at the end, storing the product C. The time delay for this stage is also 9 gate delays, i.e., 8 gate delays for the 64-bit carry select adder and one for the register R3.

Figure 3 shows the detailed data flow structure of the proposed 32-bit multiplier, in two stage of pipelining.

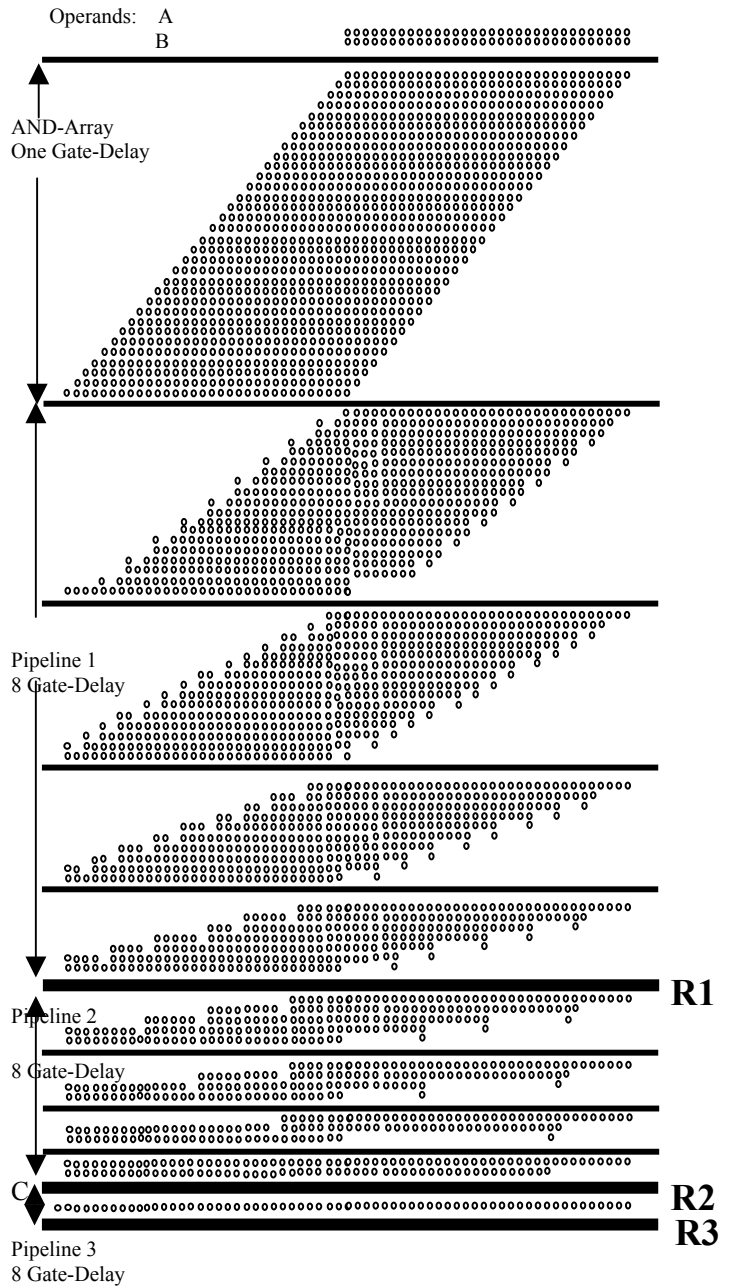


Fig.3 – The data flow structure of the 32-bit multiplier with two stage pipelining.

Notice that, although the final product is going to be maximally 64-bit long but because the two operand adder needs to start from the 10th position, rather than the first (LSB) position, we only need to use a 55-bit adder unit. This although does not alter the logarithmically reduced time delay but it offers a significant saving in the hardware area. Figure 4(a) shows the list of entries (bits) in each column and in each layer of the Wallace adders structure for the 32-bit multiplier, and Fig.4(b), on the other hand, shows the number of mixed adders, MA, used in each layer of the structure.

As discussed earlier, the number of entries from one layer in the Wallace tree to the next is reduced by a factor of 3/2. As a result, the number of bits in each column is logarithmically

reduced as we move on, from one layer to the next. Hence, the time delay associated with a Wallace structure takes a logarithmic form in relation to the word size. Table 1 shows the proportionality between the data size and the number of layers (delay) in the Wallace tree. It is important to notice that, the time delay through the Wallace structure and that through the carry select adder are both logarithmically increasing with the data size. This suggests that the design of Wallace structure, proposed for a 32-bit multiplier, is also valid for higher density data sizes without much alteration. For instance, applying the same procedure for the design of a 64-bit multiplier should result in time delay not greater than 12 gate delays.



Fig.4(a) - The Wallace tree structure for a 32-bit multiplier

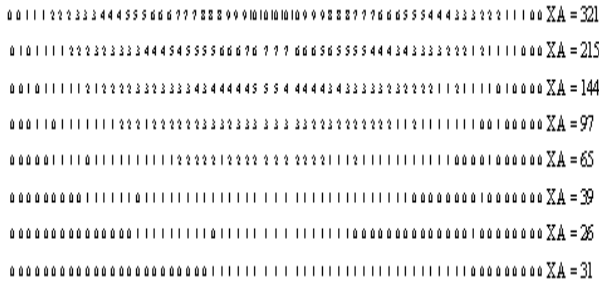


Fig.4(b) - Number of Wallace adders for a 32-bit multiplier, XA indicates the number of mixed adders.

Table 1

Data Size	Log(n/2)/Log(3/2)	# Layers in the Wallace tree
4	1.71	2
8	3.42	4
16	5.13	6
32	6.84	8
64	8.55	10

As indicated, for our particular design of a 32-bit multiplier, Table 1 shows 8 layers for the Wallace structure and 10 for a 64-bit multiplier.

III – TWO-OPERAND ADDER

The final stage in the proposed multiplier is a 64-bit carry select adder [6]. The algorithm for this adder is based on the carry-select adder technique [7], where the operands are sliced into single bits for partial additions, with both choices of 0 and 1 input carries. In an effort to reduce the carry propagation delays, a logarithmic approach is proposed that simultaneously produces carries in groups rather than individually. It is shown that, the reduction in the carry propagation delay associated

with this logarithmic approach is more significant for higher density codes. In general, to generate all carry flags, including the output carry, from m number of slices of partial adders, it takes $k = \log_2(m)$ (2-to-1 multiplexer) gate delays. For instance, for a 64-bit adder it takes 6 gate delays to generate all 64 carry flags.

To understand the algorithm consider the addition

$$S = A + B \quad (2)$$

Where, both A and B are 64-bit operands, i.e., $A = a_{63} a_{62} \dots a_1 a_0$ and $B = b_{63} b_{62} \dots b_1 b_0$. For the i th bit the partial addition is presented by:

$$\begin{aligned} m_i &= a_i \oplus b_i \\ O_i &= a_i \cap b_i \\ n_i &= \overline{a_i \oplus b_i} = \overline{m_i} \\ I_i &= a_i \cup b_i \end{aligned} \quad (3)$$

Where, m_i and O_i are the primary sum and carry terms for 0 input carry, and n_i and I_i are the primary sum and carry terms for 1 input carry, all respectively. Note that we get the entire partial sum and carry terms within one gate delay. This is because all the sums and carries are generated simultaneously.

Now, to get the final value for the sum term, S_i , as well as for the carry flag, C_i , we need to use the preceding carry flag, C_{i-1} , and make the selection, as depicted in Eq.(4).

$$\begin{aligned} S_i &= m_i \overline{C_{i-1}} + n_i C_{i-1} \\ C_i &= O_i \overline{C_{i-1}} + I_i C_{i-1} \end{aligned} \quad (4)$$

Equation (4) shows a sequentially generated carry flag scheme, where it starts from the LSB and carries are generated sequentially in ascending order. Figure 5 shows a block diagram for a 64-bit adder. As indicated, the adder consists of three circuit blocks: Partial Adder, Carry Generation Block, and Sum Block. The Partial Adder simply generates single bit partial sums and partial carries for input carries 0 and 1, as given in Eq.(3). The Carry Generation Block (channel), on the other hand, generates all carry flags, from c_1 to c_{63} . Finally, with carry flags available, the final sum term for each bit is selected within the Sum Block. The Sum Block is simply a 64-bit array of XOR gates, one for each summation bit.

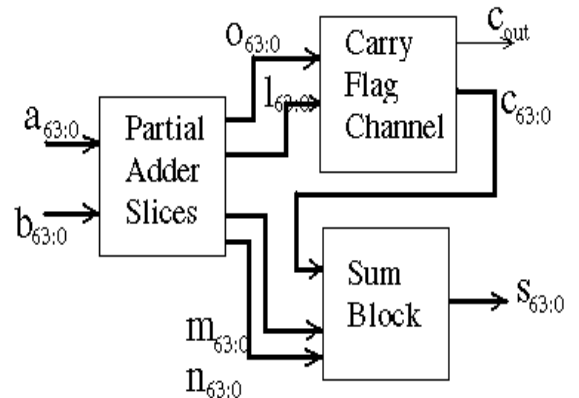


Fig.5 - A block diagram for a 64-bit carry select adder.

Table 4

Description	Pipeline sections	# Gates	# Gate-delays
AND-Array	1	1024 AND	1
Wallace – 1	1	4638	8
Register (R1)	-	252 FF	1
Wallace – 2	2	942	8
Register (R2)	-	117 FF	1
2-Operand Adder	3	604	8
Register (R3)	-	64 FF	1
32-bit Multiplier	-	7641	10

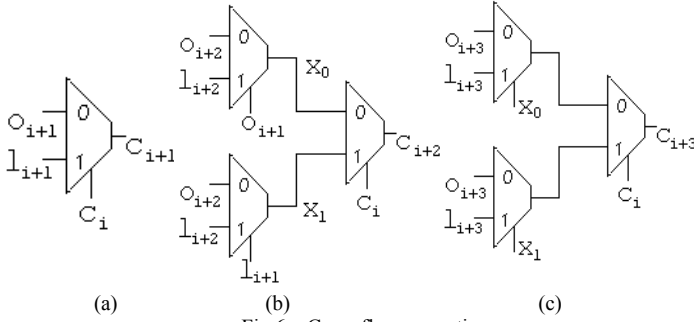


Fig.6 – Carry flag generation.

Carry Generation Block

To understand how the carry flag circuit works we refer to Fig.6(a). Here, the carry flag, c_{i+1} , is generated through a 2-to-1 MUX that selects between the two primary carry terms O_{i+1} and I_{i+1} , by using c_i . On the other hand, using c_i , as shown in Fig. 6(b) could also generate the carry flag c_{i+2} . This is done without actually adding to the normal time delay. Similarly, Fig.6(c) shows how the carry flag, c_{i+3} is directly generated from c_i . Eventually, we should be able to generate any higher carry flag directly from a lower carry flag provided that the path delay does not exceed the flag delay. It is shown [6] that logarithmic generation of higher flags is the most optimal procedure. Table 2 specifies the path delays (in scale of a gate delay) of different carry flags generated in logarithmic format.

Table 2

C_1	1 gate delay
C_2 and C_3	2 gates delay
C_4, C_5, \dots and C_7	3 gates delay
C_8, C_9, \dots and C_{15}	4 gates delay
C_{16}, C_{17}, \dots and C_{31}	5 gates delay
C_{32}, C_{33}, \dots and C_{63}	6 gates delay

Table 3 provides the gate counts for the carry generation block and per carry flag, for different size operands.

Table 3

Operand Size	Carry Generation Block Gate Counts	Gate Counts Per Carry Flag
2	2	1
4	6	1.5
8	18	2.25
16	50	3.125
32	130	4.0625
64	322	5.03125

IV – OVERALL PERFORMANCE AND CONCLUSION

Table 4 shows the number of gates and delays associated with a 32-bit multiplier. Note that with three sections pipelining (Fig. 3) we get a balance of 8-gate-delays between the sections, and overall 10 gate-delays for the product throughput.

In conclusion, the design of a high performance and high-density 32-bit multiplier is proposed. This multiplier is constructed by using the Wallace tree structure with two sections pipelining. For the two operand, produced by the Wallace structure, a fast carry select adder is implemented. As shown, the time delay for both the Wallace tree and the two-operand adder increases logarithmically with the data size. Therefore, the delay for the entire multiplier is $O(\log(n))$, n being the data size. The design is particularly carried out for a 32-bit multiplier with two sections of pipelining. This, as discussed, balance the delays through the multiplier for a high-speed throughput. It is shown that a total of ten gate delays is all needed for the multiplication throughput.

V. REFERENCES

- [1] N. R. Scott, *Computer Number Systems and Arithmetic*. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 54-57.
- [2] B. Parhami, *Computer Arithmetic, Algorithm and Hardware Design*, Oxford University Press, New York, 2000, pp. 114-123.
- [3] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electronic Computers*, vol. 13, pp. 14 – 17, 1964.
- [4] L. Dadda, "On Parallel Digital Multipliers," *Alta Frequenza*, vol. 45, pp. 574 – 580, 1976.
- [5] E. E. Swartzlander, "Parallel Counters," *IEEE Trans. Computers*, vol. 22, no. 11, pp. 1021 – 1024, 1973.
- [6] R. Hashemian, "A New Design for High Speed and High-Density Carry Select Adders", *43rd Midwest Symposium on Circuits and Systems*, Lansing, Michigan, August 8-11, 2000.
- [7] J.O. Bedrij, "Carry-Select Adder," *IRE Trans. Electronic Computers*, Vol. 11, pp. 340346, 1962.
- [8] Nigaglioni, R. H., and E. E. Swartzlander, "Variable Spanning Tree Adder," *Proc. Asilomar Conf. Signals, Svstems, and Computers*, 1995, pp. 586-590.
- [9] B. Gilchrist, J. H. Pomerence, and S. Y. Wong, "Fast carry logic for digital computers," *IRE Trans. El. Comp.*, vol. EC-4, no. 4, pp. 133-136, Dec. 1955.
- [10] H. L. Garner, "A survey of some recent contributions to computer arithmetic," *IEEE Trans. Comput.*, vol. C-25, pp. 1277-1282, 1976.
- [11] N.H.E. Weste and K. Eshraghian, *PRINCIPLES OF CMOS VLSI DESIGN*, Addison-Wesley Pub. Comp., 1993
- [12] J. Rose, A.E. Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proc. IEEE*, vol. 81, no. 7, July 1993, pp.1013-1029.