High-Speed Multiplier Design Using Multi-Input Counter and Compressor Circuits

Mayur Mehta* and Vijay Parmar
Advanced Micro Devices
5204 E. Ben White Blvd.
Austin, TX 78741

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

Abstract

Multiplication represents one of the major bottlenecks in most digital processing systems. Depending on the word-size, several partial products are added to evaluate the product. The well-known shift-and-add algorithm uses minimal hardware but has unacceptable performance for most applications. Several parallel fast multiplication schemes have been suggested using several levels of blocks containing full adders.

This paper presents the design of a fast multiplier implemented using either (7,3) parallel counter or (7:3) compressor circuits for implementation in CMOS technology. The resulting 16 by 16-bit multiplier has less delay than conventional fast multipliers, although the gate count is about 10% higher.

1 Introduction

Multiplication is inherently a slow operation as a large number of partial products are added to produce the product. For example, in a 16 by 16-bit multiplication, 16 partial products are added. Modified Booth encoding [1] can be used to reduce the number of partial products. A modified Booth encoding algorithm looking at groups of three bits at a time will reduce the number of partial products in our example from 16 to eight. That is still a large number and will involve a substantial delay in comparison with other functional units in the system such as adders. In applications like digital signal processing, this delay is unacceptable, particularly in the context of ever increasing throughput requirements. Recent Reduced Instruction Set Computing processors have also started to include multiplier units, since many applications make extensive use of multiplication. Researchers have developed several fast multiplication approaches.

Wallace [2] suggested the idea of pseudo-adders, which are essentially arrays of full adders without rippling carries, that take three inputs and reduce them to two equivalent outputs. Wallace used pseudo-adders at several levels in the summation of the partial products. Thus, for 16 by 16-bit multiplication without Booth encoding, where there are 16 partial products, the Wallace multiplier uses five pseudo-adders to reduce the 16 partial products to eleven at the first level. It uses three pseudo-adders at the second level to reduce to eight partial products, two at the next level to get six partial products, two more pseudo-adders to get four partial products, and one each at the following two levels finally producing two partial products. At this point, a fast carry propagate adder (CPA) such as a carry lookahead adder, carry skip adder, carry select adder, etc. is used to determine the final product. Thus, a Wallace multiplier has a total delay equivalent to six full adder delays plus one CPA delay. A shift-and-add algorithm has 15 CPA delays for the same case, although it uses less hardware. The array multiplier scheme used by other researchers [3], makes use of only one pseudo-adder at each level. It reduces the number of partial products by one at each level. Therefore, for the 16 by 16-bit multiplication case cited above, it takes 14 full adder delays and one CPA delay.

Dadda [4], [5] generalized the idea of using full adders to reduce the partial product matrix by introducing the concept of (n,m) parallel counters. An (n,m) parallel counter is a combinational network with n inputs and m outputs where the outputs express the count of the number of inputs that are ONEs. Thus, a full adder is a (3,2) parallel counter. Dadda aimed at reducing the height of the partial product matrix by application of suitable parallel counters. He further theorized that depending on the available parallel counters, a sequence of numbers exists which could be used to determine the appropriate height of the partial product matrix at each level. At every level, he used just enough parallel counters to reduce the height of the partial product matrix to the next lower number in the sequence. When using (3,2) parallel

* Currently with Ross Technology, Inc. Austin, TX 78736

counters, the sequence is 2, 3, 4, 6, 9, 13, 19, etc. A 16 by 16-bit multiplier using Dadda's scheme has the same delay as the Wallace multiplier and requires fewer gates, but has a less regular structure and might be more difficult to lay out in VLSI.. Dadda's scheme can be implemented with parallel counters other than (3,2) counters.

In the Dadda (n,m) parallel counter, n is the number of inputs with equal weight. The n bits must come from the same column of the partial product matrix. Stenzel, *et al.* [6] extended this idea to include parallel counters with input bits coming from multiple columns, which are described as $(c_{k-1}, c_{k-2}, \ldots, c_0, d)$ parallel counters, where k is the number of input columns, c_i is the number of input bits in the column of weight 2^i, and d is the number of output bits. Counters like (5, 5, 4), (2, 2, 2, 3, 5) and (3, 3, 3, 3, 6) were suggested and the Wallace or Dadda scheme was used to reduce the partial product matrix height to two rows. The design included a 4 by 4 array implemented with a ROM for the generation of the partial product matrix instead of the AND gate arrays used by others. This reduces the initial number of partial products. Appropriate multi-input parallel counters were used to reduce the height of the partial product matrix. The major drawback of this design was the use of ROMs for implementing the parallel counters. While a ROM based approach is practical for LSI technology, it is impractical with current VLSI technology since ROMs are slow and occupy substantial area. Multi-column parallel counters implemented using combinational logic have large delay due to the need to propagate the carry across several columns.

In implementing multipliers the word sizes are generally multiples of two. Researchers have explored this idea by using a (4:2) compressor to convert four bits to two. This block is really a (5,3) counter that takes four input bits, one intermediate carry input bit from the previous column, and generates an intermediate carry output with weight two and two output bits with weights one and two. Recently Santoro and Horowitz [7] implemented a 64 by 64 array multiplier with (4:2) compressors realized using pairs of (3,2) counters. Earlier Shen and Weinberger designed a (4:2) compressor without using (3,2) counters [8]. They used exclusive-OR gates to derive the sum bit from the five inputs. The intermediate carry is generated using the four inputs, while the carry output is based on the four inputs and the intermediate carry from the previous block. More recently, Nagamatsu, *et al.* [9] used this approach with minor modifications to implement a 32 by 32 bit multiplier using a (4:2) compressor circuit realized with 0.8-μm CMOS.

Swartzlander [10] introduced a methodology to design a counter with 2k+1 inputs, using two k input counters and a $(1+\lfloor \log_2(k) \rfloor)$ stage ripple carry adder. Using this philosophy a (7,3) counter can be designed using two (3,2) counters and a two stage ripple carry adder (which is equivalent to two (3,2) counters).

This paper, describes a multiplication scheme using a (7,3) counter circuit that offers substantial improvement over the one implemented with (3,2) counters. To illustrate its utility, the logic design of a 16 by 16-bit multiplier using (7,3) parallel counters is described. It has fewer gate delays than the Wallace and Dadda schemes for a 10% increase in gate count. It has slightly more gate delays than the (4:2) compressor based 16 by 16-bit multiplier with a slightly smaller gate count. Since the counter based design has fewer inter cell data lines, it is expected to be easier to lay out and may be faster due to the reduced capacitive loading on the interconnections. A design for a multiplier based on (7:3) compressor circuits exhibits almost identical delay and area characteristics to the based on (7,3) counters.

The next Section presents the design of the (7,3) counter. This methodology is useful for designing similar blocks with a higher number of inputs which would be suitable for large word-size multipliers. The (7,3) counter is also compared with a (7,3) counter designed using (3,2) counters. After that, a (7:3) compressor which is designed by modifying the (7,3) counter is described. Then, practical application of these circuits is demonstrated by considering a 16 by 16-bit multiplier design. Some of the decisions in the design are scalable to larger word-sizes. The design is compared with similar implementations using the Wallace, Dadda, and Nagamatsu schemes.

2 (7,3) Parallel Counter

Figure 1 shows the logic diagram of the proposed (7,3) counter. The seven inputs are divided into two groups (X_0, X_1, X_2, X_3) and (X_4, X_5, X_6). Internal signal A is the carry for two, three, or four ONEs in the first group, while internal signal B is the carry for two or three ONEs in the second group. Finally, internal signal C is generated by ORing the carry for 4 ONEs in the first group (along with A) and 1-1, 1-3, 3-1, and 3-3 interactions between the two groups. A, B, and C are combined using a conventional full adder to get outputs C_1 and C_2, with weights of two and four respectively. This circuit has been tested exhaustively through logic simulations to confirm that it correctly implements the described functionality.

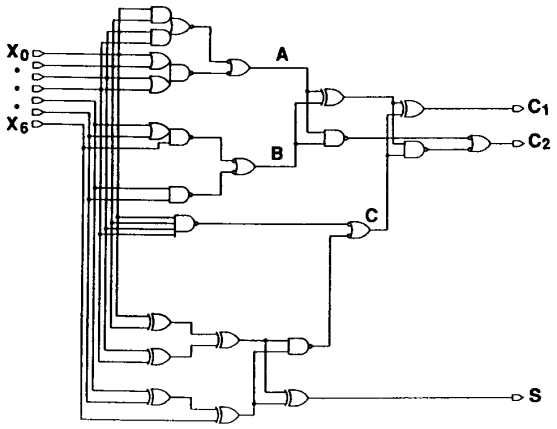

Figure 1. (7,3) Parallel Counter.

Figure 2 shows the block diagram of a (7,3) parallel counter implemented using (3,2) counters. First two (3,2) counters take six of the seven inputs and generate two sum and two carry outputs. The sum outputs are combined with the seventh input in another (3,2) counter to generate the S output of the (7,3) counter. The carry output of this (3,2) counter is combined with the carry outputs from the two first level counters using a fourth (3,2) counter to yield C_1 (which is the sum output) and C_2 (which is the carry output), with weights of two and four respectively. This circuit has six exclusive-OR delays. Assuming that an exclusive-OR gate has twice the delay of a normal gate, and assuming that complex AND-AND-NOR and OR-OR-NAND gates have 1.5 times the delay of a normal gate, the proposed (7,3) counter has four exclusive-OR delays which is a 33 percent improvement over the one using (3,2) counters while using approximately the same number of gates. The next Section presents a (7:3) compressor circuit designed by modifying the (7,3) counter.

3 (7:3) Compressor

In designing a (4:2) compressor, a problem arises since a sum output with a weight of one and a carry output with a weight of two are not enough to convey the maximum possible count of four. This problem is circumvented by generating an intermediate carry output which is fed into the next block in the adder array. Thus a (4:2) compressor is effectively a (5,3) counter. A block diagram of a (4:2) compressor is shown in Figure 3. The first box takes the four inputs and generates one sum output S' and two carry outputs C_{out} and C'. C' is generated only when S' is zero. The next box takes S', C_{in}, and C' as inputs and generates S and C. Since S and C cannot represent the case when C', S', and C_{in} are all ONEs, C' is forced to be ZERO when S' is ONE. Carries in such cases are accounted for by C_{out} which is essentially the C_{in} to the next (4:2) compressor. One advantage of this scheme is that the intermediate carry output, C_{out}, is generated only by the four inputs and is used in the following block to generate C and S. This avoids carry propagation across more than two blocks.

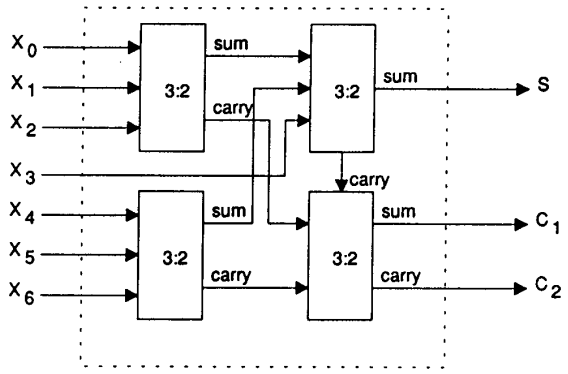

Figure 2. (7,3) Counter Implementation with (3,2) Counters.

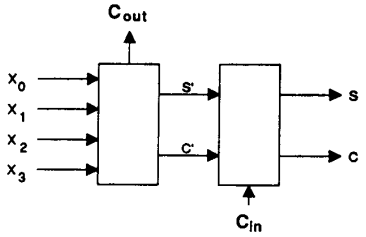

Figure 3. (4:2) Compressor.

A similar idea was used in designing the (7:3) compressor. Figure 4 shows a block diagram of this compressor. Of course, three outputs are enough to count seven bits, and the intermediate carry outputs could have been avoided. They were retained to explore the

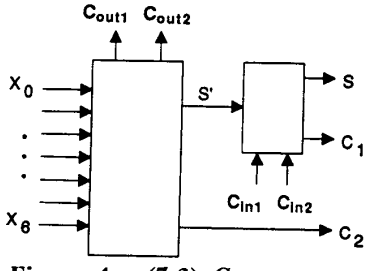

Figure 4. (7:3) Compressor.

possibilities of reducing the overall delay and gate count of the multiplier. The left box takes inputs X_0-X_6 and generates the sum output S' and carry outputs C_{out1}, C_{out2}, and C_2. C_{out1} and C_{out2} are intermediate carries which are fed into the next (7:3) compressor block in the adder array as C_{in1} and C_{in2}. In the next box, S', C_{in1}, and C_{in2} are added using a conventional (3:2) counter to generate S and C_1. S, C_1, and C_2 are the three output bits. Note that this circuit differs from the (7,3) parallel counter in that C_1 and C_2 both have a weight of two.

Figure 5 shows the logic diagram of the (7:3) compressor. This circuit is very similar to the (7,3) counter circuit. C_{out1} is generated only from X_0, X_1, X_2, and X_3 for cases when two, three, or four of them are ONEs. Similarly, C_{out2} is generated from X_4, X_5, and X_6. This is useful when this block is used with five inputs as C_{out2} is not generated in that case. Usually, the next block in such cases is a Nagamatsu (4:2) compressor which takes only one intermediate carry input. C_2 takes care of the carries not accounted by C_{out1} and C_{out2}.

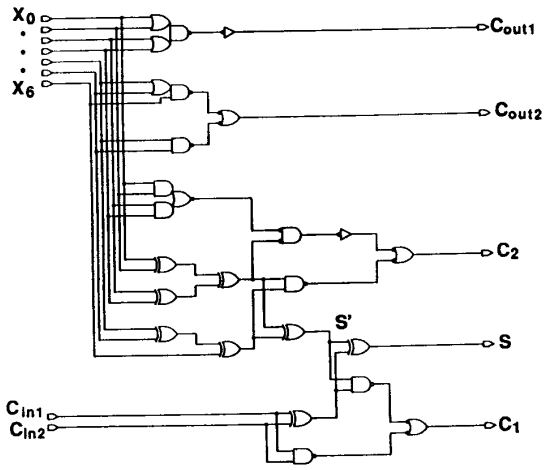

Figure 5. (7:3) Parallel Compressor.

4 Implementation

A CMOS 16 by 16-bit multiplier has been designed using the (7,3) counter and (7:3) compressor circuits described above. The multiplier uses 16-bit magnitudes, but could be modified to accept twos complement numbers by the Baugh and Wooley approach [11]. Booth encoding is not used here, although the approach is applicable to multipliers using modified Booth encoding [1]. The parallel structure closely resembles the Wallace scheme with some modifications. Figures 6 and 7 show the partial product matrices for multipliers using (7,3) counters and (7:3) compressors respectively.

The partial product matrix is generated using an array of AND gates. In the Wallace multiplication scheme, each row of the partial product matrix is input to an array of adders (compressors). As shown in Figures 6 and 7, this was modified slightly to reduce the overall delay. Thus, partial products included in the first block are the ones in the top seven rows plus some of the bit products from the two rows below. This is shown by the solid line which steps down on the left. The same procedure is repeated for the second block at the top level. As a result of this modification, four bit products are left at the lower right corner (columns 14, 15 and 16) of the partial product matrix. These are reduced to one partial product using two half adders. This results in seven partial products at second level which are reduced to three and from then to two using arrays of full adders (i.e., (3:2) counters). If this modified approach were not used, there would have been eight partial products at the second level and four at the next level. The (4:2) compression would have been required at the third level which would have added to the delay and the gate count.

Half-adders (2,2), full adders (3,2), (4,3) counters or (4:2) compressors, (5,3) counters or (5:3) compressors, and (6,3) counters or (6:3) compressors were used at the right and left end of the top two levels. (4,3), (5,3) and (6,3) counters are modified (7,3) counters with slightly reduced gate count. Similarly, (5:3) and (6:3) compressors are modified (7:3) compressors.

5 Discussion

The complexity of a variety of different implementations of a 16 by 16-bit multiplier is shown on Table 1. The Table shows the number of components (i.e., adders, compressors, and counters) and the total equivalent gate count for several multipliers. The equivalent gate count is the sum of number of "simple" gates (i.e., inverters, 2-input and 3-input NAND and NOR gates), 1.5 times the number of complex gates, and two times the number of exclusive-OR gates. It is expected that an implementation using (7,3) counters will be easier to lay out due to the absence of intermediate carries.

The multiplier delay can be compared by evaluating the number of gate delays to reduce the number of partial products from 16 to two. Note however that the total time to perform the multiplication must also include the time to generate the bit products (one gate delay) and the time to sum the two words in a carry propagate adder (ten

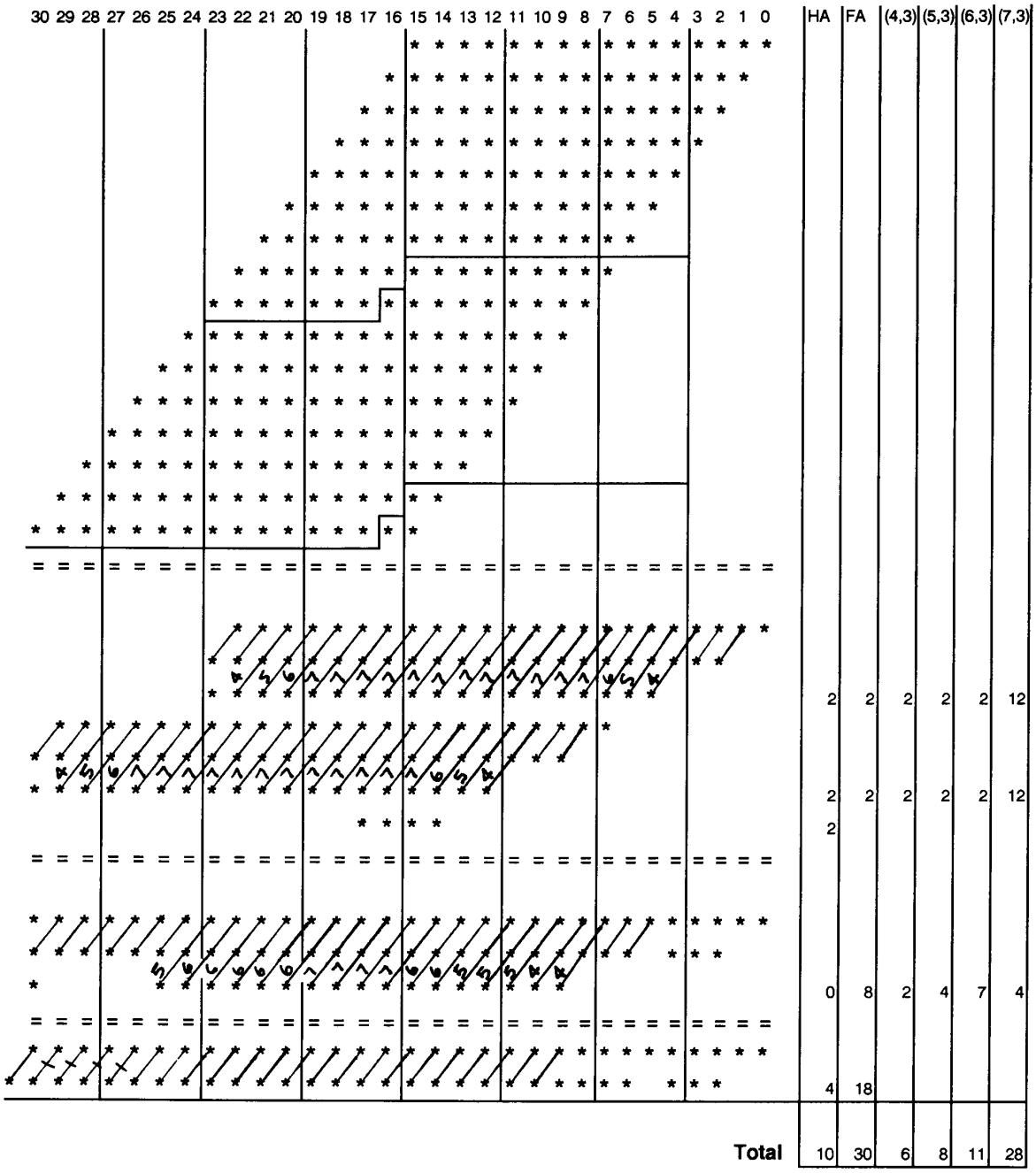

Figure 6. 16 by 16-Bit Multiplier Implemented with (7,3) Counters.

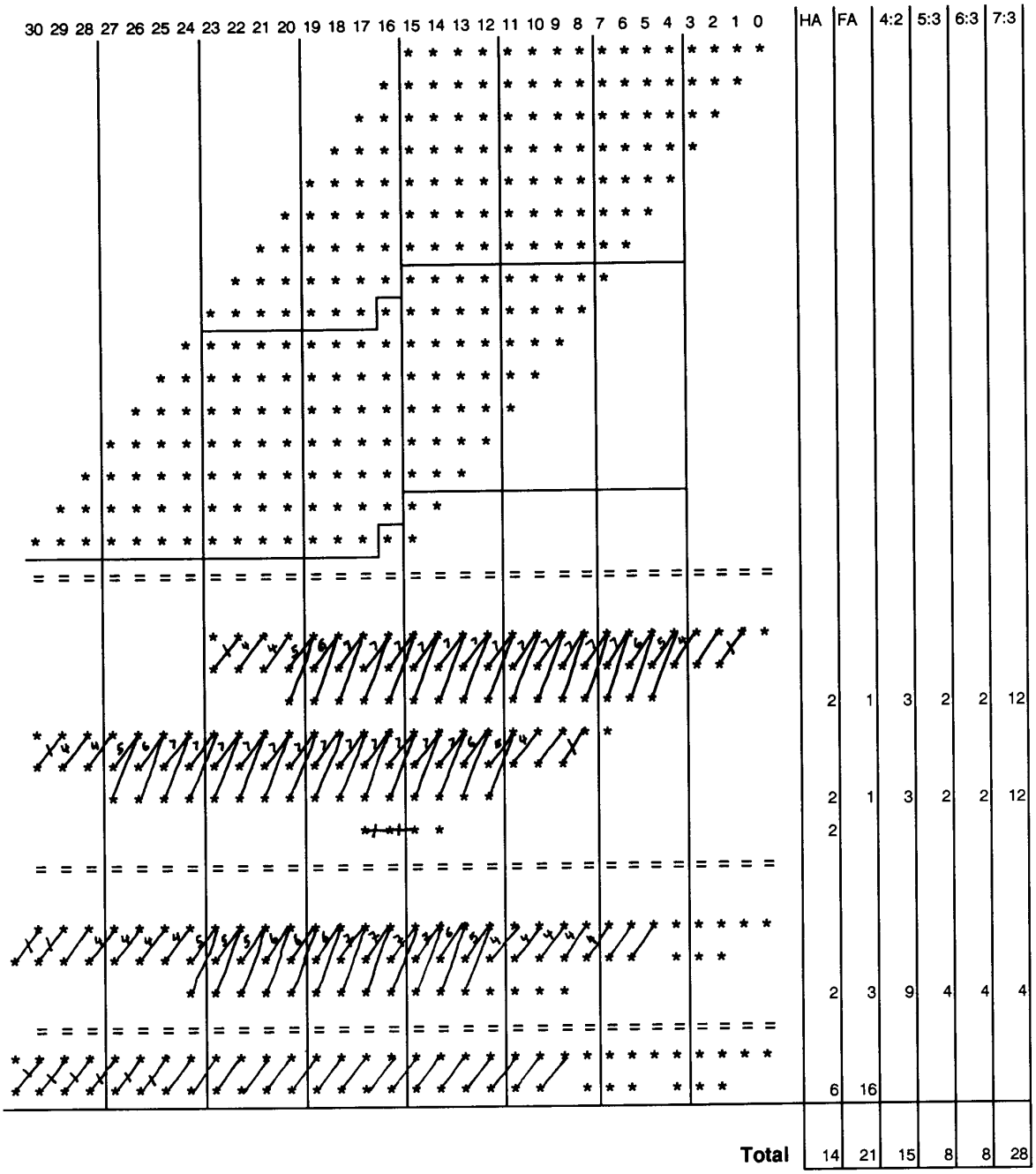

Figure 7. 16 by 16-Bit Multiplier Implemented with (7:3) Compressors.

Table 1. Comparison of the Complexity of Various 16 by 16-Bit Multipliers.

	Wallace	Dadda	(4:2) Compressor	(7:3) Compressor	(7,3) Counter
Half Adder	35	16	18	14	10
Full Adder	200	196	18	21	30
(4:2) Compressor			95	15	
(5:3) Compressor				8	
(6:3) Compressor				8	
(7:3) Compressor				28	
(4,3) Counter					6
(5,3) Counter					8
(6,3) Counter					11
(7,3) Counter					28
Total Gate Count	1575	1452	1641	1650	1623

delays for a carry lookahead adder). The Wallace and Dadda multipliers both use six levels of full adders (four gate delays each) to reduce the number of partial products from 16 to two. As each full adder has two exclusive-OR gate delays (two "simple" gate delays each), their delay is 24. The (4:2) compressor approach uses three exclusive-OR gates per level, using three levels to reduce the partial products to two giving a total of 18 gate delays. Both of the new schemes (i.e., using (7:3) compressors and using (7,3) counters) have a delay of eight exclusive-OR gate delays plus one full adder delay for a total of 20 "simple" gate delays. Implementations based on (4:2) compressors, (7,3) counters, and (7:3) compressors improve upon the Wallace and Dadda schemes in terms of overall delay.

One of the advantages of the multiplier implementation using (7,3) counters over the implementation with (4:2) compressors is the reduced number of interconnections. This has been established in IBM RS/6000 Floating Point Unit design [12] which uses (7,3) counters to implement a 56 by 56 multiplier. The (4:2) compressor reduces four bits to two at each level, while the (7,3) counter reduces seven bits to three. Clearly a multiplier implemented with (4:2) compressors will require more blocks at the early stages of the bit product reduction process than one using (7,3) counters. At lower levels, as the number of rows of partial products goes down, implementations using (4:2) compressors become more efficient. For example, a 56 by 56 bit multiplier requires eight (7,3) counter arrays at the top level to reduce the number of partial products to 24 or 14 (4:2) compressor arrays to reduce the number of partial products to 28. At the next level, it requires three (7,3) counter arrays to reduce to twelve partial products or seven (4:2) compressor arrays to reduce to 14 partial products. In addition to the increased number of interconnections because of the greater number of blocks, (4:2) compressors also require connections between adjacent compressors for intermediate carries. The major difference between implementations using (7,3) counters and (7:3) compressors is that the compressor based implementations require interconnections between adjacent blocks for the intermediate carries.

The (4:2) compressor based approach has a slightly lower delay with a gate count that is slightly higher than that of the (7,3) parallel counter implementation. Saving two gate delays out of a total of 31 (this includes one delay for the AND gates that generate the bit product matrix, 20 delays for the reduction to a two row matrix, and ten delays for a carry lookahead adder) may be less significant than the speed variation due to gate loading which is expected to be less for the (7,3) parallel counter implementation.. Multipliers implemented with (4:2) compressors are expected to be harder to lay out than those implemented with (7,3) counters as there are more interconnections.

6 Conclusion

A novel scheme for parallel multiplication using (7,3) counter circuits has been designed and discussed. A (7:3) compressor circuit was investigated as an extension of the (4:2) compressor concept. This study indicates that parallel multipliers implemented using (7,3) counters have better performance than those implemented using (7:3) compressors. They exhibit identical delay characteristics while the counter implementation requires fewer gates and lays out better. Although the (7,3) counter implementation uses more gates than the Wallace and Dadda schemes, it achieves a lower delay. The (7,3) counter based implementation compares favorably with the (4:2) compressor implementation in terms of gate count, although it has slightly higher delay for the 16 by 16-bit multiplier example.

References

[1] H. Sam and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementations," *IEEE Transactions on Computers*, Vol. 39, pp. 1006-1015, 1990.

[2] C. S. Wallace, "A suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, Vol. EC-13, pp. 14-17, 1964.

[3] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, New York: Wiley, 1979.

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

[5] L. Dadda, "On Parallel Digital Multipliers," *Alta Frequenza*, Vol. 45, pp. 574-580, 1976.

[6] W. J. Stenzel, W. J. Kubitz, and G. H. Garcia, "A Compact High-Speed Parallel Multiplication Scheme," *IEEE Transactions on Computers*, Vol. C-26, pp. 948-957, 1977.

[7] M. R. Santoro and M. A. Horowitz, "SPIM: A Pipelined 64 x 64-bit Iterative Multiplier," *IEEE Journal of Solid-State Circuits*, Vol. 24, pp. 487-493, 1989.

[8] D. T. Shen and A. Weinberger, "4-2 Carry-Save Adder Implementation Using Send Circuits," *IBM Technical Disclosure Bulletin*, Vol 20, pp. 3594-3597, 1978.

[9] M. Nagamatsu, *et. al.*, "A 15-ns 32 x 32-b CMOS Multiplier with an Improved Parallel Structure," *IEEE Journal of Solid-State Circuits*, Vol. 25, pp. 494-497, 1990.

[10] E. E. Swartzlander, Jr., "Parallel Counters," *IEEE Transactions on Computers*, Vol. C-22, pp. 1021-1024, 1973.

[11] C. R. Baugh and B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, Vol. C-22, pp. 1045-1047, 1973.

[12] R. K. Montoye, E Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 Floating-Point Execution Unit," *IBM Journal of Research and Development*, Vol. 34, pp. 59-70, 1990.