A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach

Vojin G. Oklobdzija, *Fellow, IEEE,* David Villeger, and Simon S. Liu

Abstract—This paper presents a method and an algorithm for generation of a parallel multiplier, which is optimized for speed. This method is applicable to any multiplier size and adaptable to any technology for which speed parameters are known. Most importantly, it is easy to incorporate this method in silicon compilation or logic synthesis tools. The parallel multiplier produced by the proposed method outperforms other schemes used for comparison in our experiment. It uses the minimal number of cells in the partial product reduction tree. These findings are tested on design examples simulated in 1μ CMOS ASIC technology.

Index Terms—Parallel multiplier, partial product reduction, Wallace tree, Dadda's counter, VLSI arithmetic, Booth encoding, 3:2 counter, 4:2 adder, array multiplier.

1 Introduction

THE increased level of integration brought by modern VLSI and ULSI technology has rendered possible the integration of many components that were considered complex and were implemented only partially or not at all in the past. The multiplication operation is certainly present in many parts of a digital system or digital computer, most notably in signal processing, graphics and scientific computation. Therefore, it became quite common to see a multiplier implemented in full in many parts where it was not found before. Examples of such are floating-point processors and recently graphics processor, various kinds of digital signal processors used for user interfaces, communication or code compression. Parallel multipliers have even migrated into the fixed-point processor of digital computers for the purpose of speeding up and facilitating address calculation needed for fast and efficient indexing through arrays of data. The speed of the parallel multiplier has always been a critical issue and, therefore, the subject of many research projects and papers [1], [2].

Several popular and well-known schemes, with the objective of improving the speed of the parallel multiplier, have been developed in the past. The first departure from the iterative array structure [3] has been described in a paper by Wallace [7]. Wallace has introduced a notion of a carry-save tree constructed from one-bit Full Adders as a way of reducing the number of partial product bits in a fast and efficient way. The notion of counters and a generalization of the Wallace scheme have been described in the paper by Dadda [8] who also proposed a method that minimized the number of counters in a compression tree. A good survey of several possible schemes based on Dadda's method can be found in the paper by Stenzel [9].

In 1981, Weinberger [10] introduced a 4:2 compressor as a way of reducing the bits in the parallel multiplier array. His compressor was used by Santoro [12], Nagamatsu et al. [13], and Mori et al. [14]. The introduction of the 4:2 compressor (as an alternative to counters) was a departure from the traditional path which resulted in an improvement over the traditionally used Wallace and Dadda's scheme.

The use of larger compressors and families of compressors was explored by Song and DeMichelli [15]. They have also developed a 9:2 compressor and made a comparison with respect to their implementation and layout.

Use of higher order compressors yielded mixed results in some cases, however, it showed a general trend toward building compressors of larger sizes as a way of making incremental improvements in multiplier speed. In our research we took the approach of generating the compressors of maximal possible size, (i.e., the size of the multiplier) which yielded better results than the use of any previous compressor or counter family. Therefore, we gradually abandoned the notion of levels and undertook a design of an optimized one-level compressor which evolved into an optimization process involving the entire array.

Our method is based on the fact that not all inputs and outputs from a device used as a compressor (or counter) contribute equally to the delay. Therefore, we sort them in a way which favors the use of fast inputs and outputs in the paths that are critical to the speed while we assign slow inputs to the signal paths which belong to the domain where an increase in the delay is tolerable. In the creation process we examine the entire multiplier array and all the signals that are entering the compressor which lead to a

- *V.G. Oklobdzija is with Integration, Berkeley, California, and with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616. E-mail: vojin@ece.ucdavis.edu.*
- *D. Villeger is with Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique, 93162 Noisy le Grand Cedex, France*
- *S.S. Liu is with Advanced Micro Devices, Sunnyvale, CA 94088-3453.*

Manuscript received Oct. 21, 1993; revised May 19, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C96005.

global rather than local optimization. This is characterized by the use of a particular compressor optimized for a minimal delay with respect to its inputs and outputs only. Our method also leads to a general solution for any chosen device (compressor or counter). By being used interactively in conjunction with improvement of a basic compressor device, this leads to an array which is optimized for speed down to the device level.

Although Booth recoding [4], [5] is widely used in parallel multipliers, it does not change the structure of the reduction tree. Moreover, its efficiency has been denied by several authors which was consistent with our findings [6], [11, [15]. As it has been shown in [6], one row of 4:2 compressors has the same effect as Booth encoding, which is achieved in less time. Given that the method presented here achieves this compression in even less time than the one using 4:2 compressors, the use of Booth encoding in parallel multipliers ceases to be advantageous for the range of technologies discussed in this paper.

We very recently became aware of an effort to build multipliers automatically via a program which attempts to compensate for the different speed of different paths in the multiplier by by-passing some stages [16]. While the approach taken is correct, which basically abandons the idea of counters or compressors, it does not take into account that different inputs (not only outputs) have different contribution to speed, as does our method. Unlike our method, half adders are used extensively in [16] which leads to worse results in terms of speed and cell number. Moreover, that method is less general and does not allow extension to device optimization. Similar approaches have also been applied in the process of "timing optimization" used in "logic synthesis," most notably in [28] and [29].

In the last section of this paper, we consider the final carry-propagate adder used to sum the partial products which have been reduced to two rows. The Final Adder delay is an addendum to the multiplier delay and, therefore, is critical. The approaches taken in designing the Final Adder were mainly concentrated on the raw speed of the adder and did not consider the specifics related to the uneven signal arrival profile of its inputs. Our selection of the Final Adder is based on these specifics. The structure of the Final Adder is *tuned* into the signal arrival profile so that the delay of the adder is minimized under these conditions which are specific to its application. We conclude that *tuning* of the Final Adder is more important than the its raw speed and that any application of a complex and hardware consuming scheme which is not optimized with respect to the signal arrival, would only constitute a waste of resources.

Finally, in the examples of various multipliers designed in 1.0μ CMOS ASIC technology, we show the advantages of our method and how it compares to the others.

2 Comparison of Different Partial Product Reduction Schemes

2.1 Wallace and Dadda Schemes

A major departure from iterative array realization of parallel multipliers has been introduced in papers by Wallace [7] and Dadda [8]. Common to both is the use of Carry-Save form in a compression tree in order to reduce the number of partial product bits to two rows. The advantage of trees is that their speed increases with the *log* of the operand length, while this increase is linear in the case of iterative arrays. Dadda introduced the notion of *counters* and a methodology of designing trees that are optimized in terms of cell number. Thus, Dadda has treated the bit reduction process as a number of *levels* (steps), the application of which results in the reduction of the number of rows of partial products. At every level the partial product bits are passed through devices designated as *counters*, the purpose of which has been the reduction of the number of rows after a pass through a level. A (p, q) counter is defined as a combinatorial network that determines the number of ones (active signals) among its p inputs producing a result on its q bit output in the form of a binary number (count). Essential to the counter is a process of summation of the input bits. The number of output bits must be sufficient to represent all the possible sums of n bits: $2^q - 1 \geq p$. A Full Adder can be treated as a (3, 2) counter, leading to a representation of the Wallace tree as a special case of Dadda. It would be ideal if it were possible to develop a higher order counter such as (7, 3) or (15, 4) with a speed that surpasses the speed of the same counters built by using Full Adders. There were several studies undertaken in the past, most notably by Stenzel [9], however, the use of a Full Adder as a (3, 2) counter is still the most prevailing.

The process applied by Wallace and Dadda can be summarized as follows: after generating the partial products, a set of counters reduces the partial product matrix but does not propagate the carries. The resulting matrix is composed of the sums and the carries of the counters. Another set of counters then reduces this new matrix and so on, until a two-row matrix is generated. Those two rows are summed up with a Final Adder to which we will refer to as a Carry Propagate Adder (CPA). This method takes advantage of the carry save form to avoid the carry propagation until the Final Adder. In this scheme the number of levels is crucial and will determine the speed of the multiplier.

2.2 Use of 4:2 and Higher Order Compressor

A major departure from the Wallace-Dadda scheme has been the introduction of a 4:2 compressor by Weinberger of IBM [10]. His idea was made visible by Santoro who used it in his PhD thesis [11] and by Mori et al. [14] who later implemented it. The 4:2 compressor, as shown in Fig. 3, made a major difference in the way cells are interconnected by introducing a *horizontal* path and, therefore, a limited propagation of the carry signal in the multiplier. Indeed 4:2 structure is not a counter, since two output bits cannot represent five possible sums of 4 bits. Thus, a carry out is necessary and subsequently a carry in. However, since the carry out is not dependent on the carry in, only a limited carry propagation occurs. There were several benefits from using a 4:2 compressor, such as simpler and more regular wiring of the multiplier tree. More notable is the introduction of the notion of a *horizontal* and a *vertical* signal path. Naturally application of 4:2 compressors led to faster realization of a multiplier. The existence of a *horizontal* path has

led to the idea of moving the *critical path* of the multiplier tree away from the center toward the most significant end where the depth of the column of partial product bits is smaller than in the middle [18]. The exploration of this idea has brought some mixed results, mainly because a major redesign of the compressors was necessary. However, it has shed enough light on the real problem of the reduction of the partial product bits.

Naturally, researchers tried to explore the idea of 4:2 compressors further and research in this direction, most notably by Song and DeMichelli [15] led to the introduction of the 6:2 and 9:2 family. The term family is being used because the new compressors introduced were simply built on 4:2 compressors and (3, 2) counters. For example, the 9:2 compressor consists of one 6:2 compressor and three (3, 2) counters, where in turn, the 6:2 compressor contains one 4:2 compressor and two (3, 2) counters. Nevertheless, a multiplier built using 9:2 compressors showed a speed advantage over one built using 4:2 compressors.

The speed comparison for three different multipliers using, respectively, (3, 2) counters, 4:2 and 9:2 compressors for the multiplier sizes ranging from 0 to 100 bits in equivalent XOR delays in the *critical* path is shown in Fig. 1. We redesigned the 4:2 and 9:2 compressors at the gate level in order to obtain the best possible performance. The difference in speed favors 9:2 over 4:2 and (3, 2). These results are in agreement with findings in [11], [13], and [15]. We use a normalized XOR delay as a measure of speed for a particular implementation of an algorithm or a method. There are several reasons for doing this:

1) the critical path in the multiplier consists of a path through a series of XOR gates independent the algorithm is used.
2) the speed comparison is made independent on the technology, which is characterized by its ability to realize a fast XOR function.

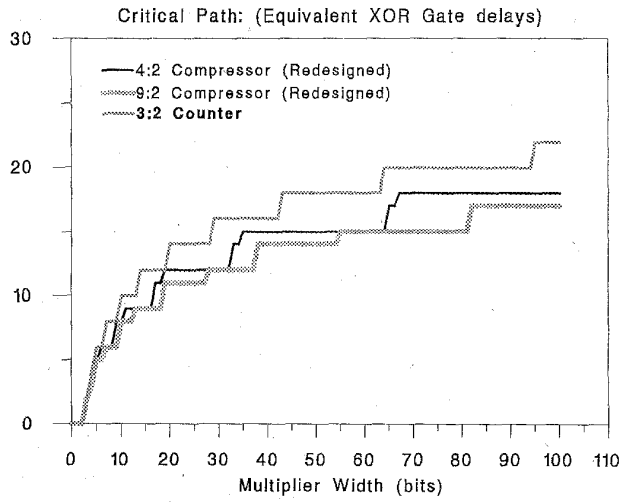


Fig. 1. Speed comparison for three different compressors (counters) used for partial product reduction: 9:2 is the best, followed by 4:2, and (3, 2).

In our early work, we followed this idea and came to the conclusion that the most efficient way to build a compressor is to start with the size of the multiplier (N), in other words to use the largest applicable size [18], [20]. The multipliers built in such a way resulted in better speed than those for which compressors of a smaller size were used [17], [18]. Though we could not claim that compressors built in such a way were optimal and would lead to the fastest realization, we identified the path to the solution. This path led us to the point where we were able to realize that the real problem is not in application or existence of a different counter or a compressor, but in the way the compressor tree has been interconnected. As we will see in the next section, any compressor can be designed with Full Adders with the speed of one designed at the gate level. Simply speaking, inside of each applied compressor, there is a Full Adder used as a building block. If that is the case, then a question arises to what is the difference? The answer is that the difference in using 4:2 and higher order compressors is not in the structure of the compressor but in the way they were interconnected.

2.3 Unequal Delays in a Full Adder: Existence of Fast Inputs and Fast Outputs

It is known that the delay from an input to an output in a Full Adder is not the same. This delay is even dependent on a particular transition (0-to-1, 1-to-0). It is also possible to come up with different realizations of a Full Adder where a specific signal path is favored with respect to the others and is designed in such a way that a signal propagation of this path takes a minimal amount of time. This is sometimes done even at the expense of other possible signal paths. For example, a ripple carry adder is designed so that the carry-in to carry-out delay is minimized. In that respect let us analyze a particular implementation of a (3, 2) counter as shown in Fig. 2. This particular case is taken from LSI 100K 1μ CMOS-ASIC cell [30]. It is used only for the illustration of the algorithm. In the case of a parallel multiplier, our design objective would be to minimize the delay from the *Input s* to the *Sum*, of the Full Adder which has direct effect on the *critical path*.

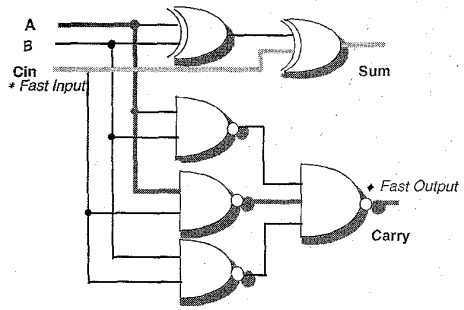


Fig.2. Signal delays in a Full Adder ((3, 2) counter).

In this example, the delay from input A or B to the Sum is equal to two equivalent XOR delays. The delay for the path from Cin to the output Sum is equal to one XOR

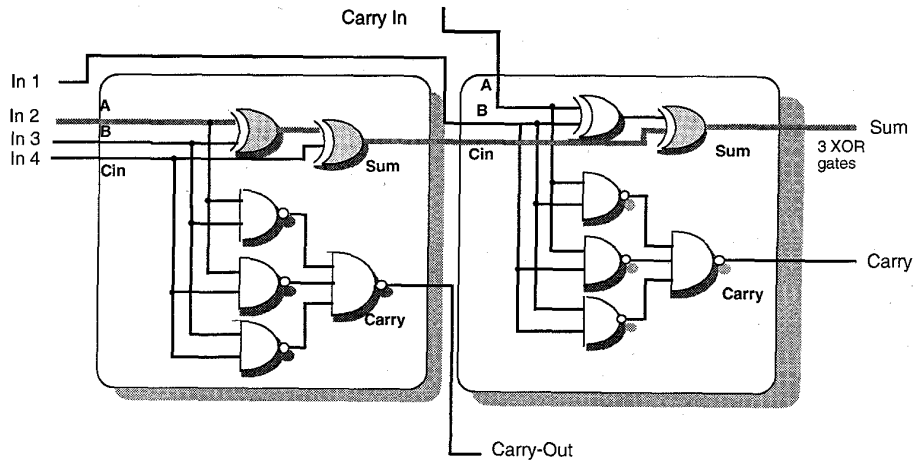


Fig. 3. Modified 4:2 compressor obtained by optimally interconnecting two Full Adders with fast input and output.

equivalent delay. We define Cin as a *fast input*. For this case, the propagation delay from A or B to the output Sum is twice as long as the propagation delay from input Cin to the Sum output. Considering the delay at the output Sum, in this particular technology delay from input A or B to Sum output is equivalent to two XOR delays. However, delay from inputs (A, B, or Cin) to the output Carry is equivalent to one XOR delay. We define Carry as a *fast output*. The value of those delays varies with technology and particular circuit implementation. In general we can use any (noninteger) values in our algorithm.

If we were to construct a 4:2 compressor by simply stacking two Full Adders together, as done by Santoro [11] the *critical path* of such a counter would be equal to four equivalent XOR delays. The researchers from Toshiba have simply redesigned the entire 4:2 compressor and treated it as a single cell. Their design resulted in three equivalent XOR gate delays and, therefore, they claimed 25% speed improvement over a conventional Full Adder Wallace tree realization.

In the next section, we will show how with proper interconnection the same speed can be achieved by using two regular Full Adders.

2.4 Improved 4:2 Compressor with Optimized Interconnections

The advantage of proper interconnection of *fast inputs* and *fast outputs* can be illustrated in the example of a 4:2 compressor. Fig. 3. shows an optimized 4:2 cell which is a result of applying our delay model and properly interconnecting *fast inputs* and *fast outputs* with the objective of minimizing the *critical path* of the 4:2 compressor. In our case, the delay through a 4:2 compressor level is equivalent to three XOR gate delays regardless of the path. If used to reduce the partial product bits in an 24 × 24-bit multiplier, the application of this modified 4:2 compressor would result in a delay of 12 equivalent XOR gates versus 16 if a regular 4:2 compressor, as used by Santoro, were applied (this numbers become 11 and 14, respectively, if a level of Full Adders is used every time the column size is three).

The use of the 4:2 compressor permits the reduction of the vertical critical path while the path involving the carry propagation, that we call *horizontal* path, is not changed. However, the *horizontal* propagation is fast and limited to 1 bit per level.

3 Three Dimensional Reduction Method

3.1 A New Approach

Instead of developing an efficient compressor structure and then using it in the process of partial product reduction, we took a global approach. In our method, we treat the entire multiplier array as a whole. The compressor consists of a vertical slice where the partial product bit array is represented in space and time (reduction steps). The vertical cross section in our representation (Fig. 4.), represents the partial product compressor of the multiplier array. The vertical slice in this representation is further interleaved with the compressors that are used in the reduction process and, therefore, represents a compressor structure corresponding to the appropriate bit position. We will refer to it as a Vertical Compressor Slice or VCS. It should be noted that there are a number of input signals into the vertical slice and a number of output signals originating from the particular vertical slice which are then being passed to the next VCS corresponding to the first higher order bit position.

Considering just one VCS we can see how the matrix of partial products is reduced by a tree of counters (Full Adders shown in the Fig. 4). However, every Full Adder produces a carry out which affects the slice of superior weight. Thus, the critical path is not only a *vertical* path through a given slice, but is also a *horizontal* path through the slices. As previously shown, the 4:2 compressor shortens the vertical path while including the horizontal path. The goal of the following scheme is to minimize both paths by building vertical slices that are optimized for a minimal delay. A method of designing VCS by considering both *vertical* and *horizontal* critical paths will be discussed.

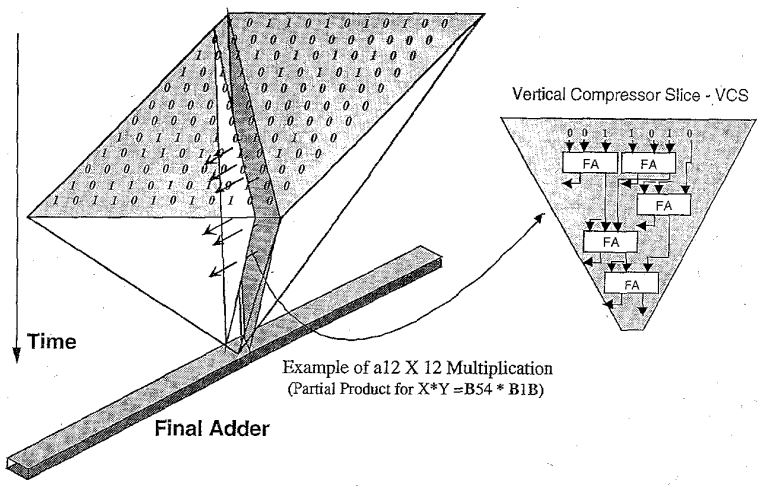


Fig. 4. A three-dimensional view of partial product reduction.

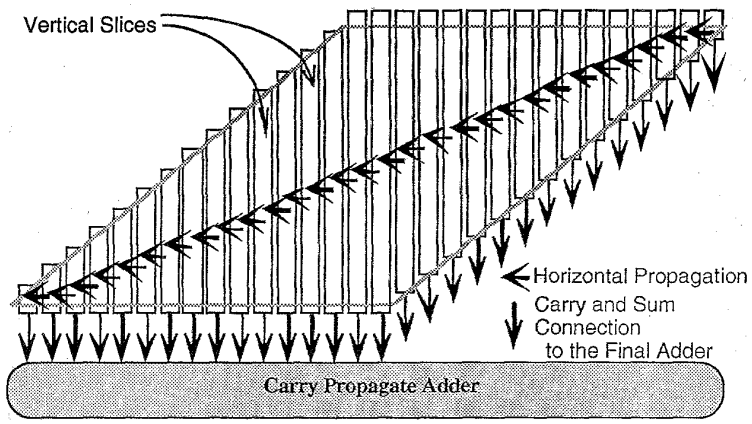


Fig. 5. The partial products matrix is divided into vertical compression slices.

The reduction of partial product bits in this new scheme is performed by creating a bit slice compressor VCS whose size is equal to the size of a given vertical cross-section of the partial product matrix, and then by assembling those compressors into an integral structure, as illustrated in Fig. 5. Since VCS are optimized by taking the neighboring VCS and their signals into account, as a contiguous process, the optimization is truly a three dimensional optimization process. We will refer to it as a Three Dimensional Minimization (TDM).

This approach is very different from Dadda's since all the partial products are compressed into a single step. This means that no intermediate partial products are considered in our case. However, TDM still produces a carry save number to be translated to a conventional form with a fast carry propagate adder (CPA). Indeed, each VCS reduces the partial products to a Sum and a Carry. Since it does not use the carry save form for the intermediate partial products, this scheme involves a carry propagation through the VCS. Therefore, it is necessary to design the VCS such that this propagation does not introduce a long delay. The main concern becomes the minimization of the *vertical* and *horizontal* critical paths rather than the number of Full Adder levels.

In addition, this scheme simplifies the design of the multiplier and its description in a hardware description language (VHDL). It also leads to an automatic generation of the partial product array by producing a net list of its signals and components. However, its efficiency will depend on the features of the VCS. The next section introduces a method of designing the VCS and subsequently the whole tree with Full Adders and half adders. This method, which automatically generates a net-list of the partial product array has been implemented in C language.

3.2 Method

The basic idea of this method is to make proper connections globally so that the delay throughout each path is approximately the same. The long delay path originating from the previous compressor should be connected to the short

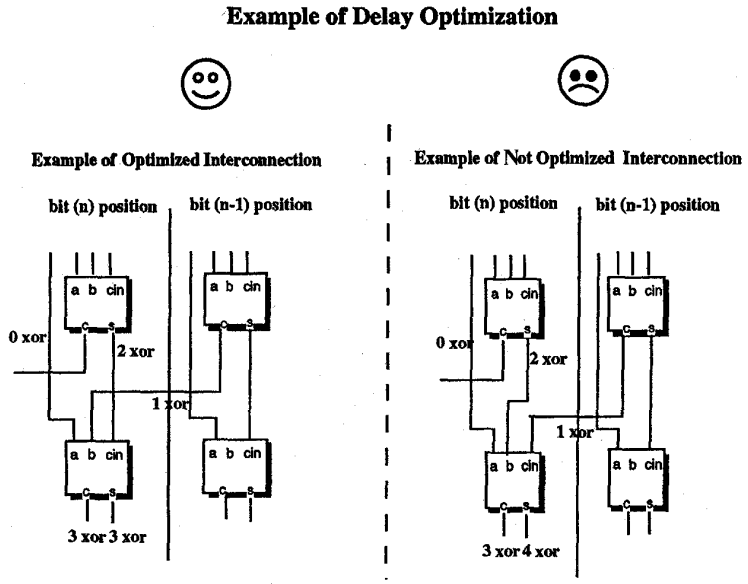


Fig. 6. Example of delay minimization by proper ordering of the input signals.

delay path of the next one, and so on. In general, this is not always possible since each output function has its unique characteristics and requires specific logic cells in its path. It is feasible to apply this idea to the partial product array using Full Adders (FA) since all of the partial product bits in the same bit position are logically the same and, therefore, interchangeable. In other words, all of the signals in any bit position can be interchanged no matter where they are originally coming from (as inputs in the same bit position, or as carries from a lower bit position). An example of how speed improvement can be achieved by application of this principle is shown in Fig. 6. The picture represents a small section of a multiplier tree. The application of this same principle resulted in the optimized 4:2 compressor shown in Fig. 3.

The presented method first creates a data structure consisting of $2N - 1$ lists L_i containing names and delays of partial products. Each VCS is represented by a list of pairs $<d_j, n_j>_i$ containing delay and name information of a node. Initially, L_i represents the inputs (names and delays) of the corresponding VCS. Consequently, its length is the number of partial products from the corresponding slice and the delays are the delays produced by the partial product generator. Initially, we assume that all of the partial product bits are generated at the same time and we chose this to be a reference point by initializing all d_j to 0. If the partial products do not arrive at the same time, d_j will be assigned corresponding delay values. Some partial products do not need to be summed in the tree, and they are directly connected to the CPA.

After sorting the elements of L_i in ascending order by the values of the delays contained in the records of the list, a FA is connected to the first three nodes of L_i. The third node, that is the slowest one, is connected to the fast input (Carry-In) of the FA. The delays of the Sum and Carry are calculated and two new pairs $<d_j, n_j>_i$ are created containing information about those signals. The pair concerning the Sum signal is inserted in L_i while the one concerning the Carry signal is inserted in L_{i+1}. The size of L_i and L_{i+1} are then adjusted. This same procedure can be applied for any general type of (p, q) compressor cell used. The use of such a (p, q) compressor is advantageous only if such a cell shows speed advantages over FA in the particular technology of implementation.

The process stops when the size of L_i reaches three. The last three signals are then connected to a FA whose signals feed the CPA. Fig. 7 illustrates the effect of this method on different arrangements of signals that have different delays.

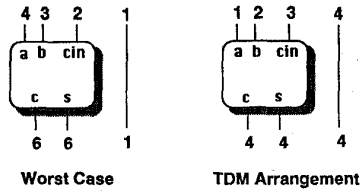


Fig. 7. Delay improvement with a different signal arrangement.

The delays of the Sum and Carry signal of the FA are calculated with the following equations:

$$\text{Delay}(S) = \text{MAX}\{\text{Delay}(A) + D_{A-S}, \text{Delay}(B) + D_{B-S}, \text{Delay}(C_{in}) + D_{Cin-S}\}$$

$$\text{Delay}(C) = \text{MAX}\{\text{Delay}(A) + D_{A-C}, \text{Delay}(B) + D_{B-C}, \text{Delay}(C_{in}) + D_{Cin-C}\}$$

where Delay(X) represents the delay attached to the signal X and the constants D_{U-V} represent the delays of a

signal crossing a FA from u to v. It should be noted that delays D_{u-v} can have any value. Those values are determined by the technology of implementation and circuit techniques used.

It is not always possible to use only FA in our design. In some cases, the use of a HA is necessary. The following demonstration shows that this use depends on the number of inputs of a VCS.

Let I_{CN} be the number of signals resulting from the application of CN counters. I_0 is the number of inputs of a VCS. Consequently, I_0 is the number of partial products at this particular bit position plus the number of carry signals originating from the previous VCS. It is not difficult to show that $I_{CN} = I_0 - CNR$ where R is defined as the difference between the number of inputs of the compressor and the number of Sum signals. In our case using FAs, R = 2 and $I_{CN} = 1$ (representing the Sum signal from the VCS). Then, $CN = (I_0 - 1)/2$. Since CN must be an integer, this expression is true if I_0 is odd. If I_0 is even, the number of Full Adders used to produce two signals is $CN = (I_0 - 2)/2$ which is an integer where the remaining two signals are reduced using a HA to produce a Sum and a Carry. Therefore, the number of cells can be expressed as:

$$CN_i = \lfloor I_i/2 \rfloor$$

In other words, a HA will be used if the number of inputs of a VCS is even. This HA is positioned near the partial product generator because carries that are generated at this position propagate through several slices. By taking advantage of the small carry delay generated by a HA, this positioning results in the gain of one XOR delay in the critical path of the multiplier.

The method, which generates the parallel multiplier bit compression array structure, is presented below. This method can be used as a basis for a program which generates a logic file containing the interconnection list as was done in our case. Such a program can be easily integrated into a silicon compiler or logic synthesis tool used for automatic generation of fast and efficient multiplier structure of any size.

3.3 Algorithm for Automatic Generation of Partial Product Array

```
Initialize:
  Form 2N - 1 lists L_i (i = 0, 2N - 2) each
    consisting of p_i elements where:
  p_i = I + 1 for i ≤ N - 1 and p_i = 2N - 1 - i
    for i ≥ N
  An element of a list L_i (j = 0, ..., p_i-1) is
    a pair: <d_j, n_j>_i where:
  nj : is a unique node identifying name
  d_j : is a delay associated with that node
    representing a delay of a signal arriving
    to the node n_j with respect to some
    reference point.
  For i = 0, 1 and 2N - 2: connect nodes from
    the corresponding lists L_i directly to
    the CPA;
For I = 2 to I = 2N - 3 {Partial Product
  Array Generation}
  Begin For
    if length of L_i is even Then
      Begin If
        sort the elements of L_i in
          ascending order by the values of
          delay d_j;
        connect an HA to the first 2
          elements of L_i starting with the
          slowest input;
            Ds = max {d_A+d_A-s, d_B+d_B-s}
            Dc = max {d_A+d_A-c, d_B+d_B-c}
        remove 2 elements from L_i;
        insert the pair <Ds, NetName> into L_i;
        insert the pair <Dc, NetName> into L_i+1;
        decrement the length of L_i;
        increment the length of L_i+1;
      End If;
    while length of L_i > 3
      Begin While
        sort the elements of L_i in ascending
          order by the values of delay d_j;
        connect an FA to the first 3 elements
          of L_i starting with the slowest
          input of the FA:
            Ds = max {dc_A+dc_A-s, dc_B+dc_B-s,
              dc_Ci+dc_Ci-s};
            Dc = max {dc_A+dc_A-c, dc_B+dc_B-c,
              dc_Ci+dc_Ci-c};
        remove 3 elements from L_i;
        insert the pair <Ds, NetName> into L_i;
        insert the pair <Dc, NetName> into L_i+1;
        subtract 2 from the length of L_i;
        increment the length of L_i+1;
      End While;
    sort the elements of L_i;
    connect an FA to the last 3 nodes of L_i;
    connect the S and C to the bit i and I + 1
      of the CPA;
  End For;
End Method;
```

The delay constants are technology dependent and are defined by the user.

3.4 Discussion of the Algorithm

The presented algorithm takes into account any delay value Ds and Dc as determined for the particular counter structure and technology of implementation. An average short wire delay and the effect of *loading* is included into Ds and Dc and calculated as a part of

$$Ds = \max \{dc_A + dc_{A-s}, dc_B + dc_{B-s}, dc_{Ci} + dc_{Ci-s}\}$$

and

$$Dc = \max \{dc_A + dc_{A-c}, dc_B + dc_{B-c}, dc_{Ci} + dc_{Ci-c}\}$$

expressions. It would be easy to generalize the same algorithm for use of higher order compressors (p, q). In such a

case, we calculate the delay D_i (i = 1, ... q) as:

$$D_i = \max \{d_j + d_{j-i} \text{ for } j = 1 \text{ to } p\}$$

Once the delays were calculated and list L_i has been sorted, we proceed by eliminating p elements from the list L_i and adding q delays to it. The objective of the algorithm is to minimize the delay of the signals that go to the next VCS. The critical path in the multiplier is usually found to be either in the largest VCS (in the middle of the multiplier tree), or as a path that starts at the least significant VCS and is being passed to the next until it ends up in the middle VCS. Therefore, our objective is to minimize:

1) the direct vertical path in a VCS (signals that are passed vertically to the FA),
2) the delay and the number of signals that are passed from VCS_i to VCS_{i+1}.

The problem of finding the multiplier structure with the minimal delay using arbitrary (p, q) counter is an NP hard problem, as is for finding a minimum delay wire layout. In the particular case presented in this paper, the problem is currently not known to be NP hard or polynomially solvable. Resolving this question is an interesting problem, and indeed we were able to prove that our algorithm produces an optimal and the best achievable structure as far as the speed of the multiplier is concerned [34].

The algorithm can easily be implemented to run in $O(n^2 \log n)$ time using priority queues, and in $O(n^2)$ time if the delays are small integers. We have not found a counter example which would yield better results for the cases of N < 64.

3.5 Example

In our case, the delays are :

$$FA_{A\to S} = FA_{B\to S} = 2 \text{ XOR delays}$$

$$FA_{Cin\to S} = FA_{A\to C} = FA_{B\to C} = FA_{Cin\to C} = 1 \text{ XOR delay.}$$

In the case of a HA, the delays become :

$$HA_{A\to S} = HA_{B\to S} = 1 \text{ XOR}$$
$$\text{delay while } HA_{A\to C} = HA_{B\to C} = 0.5 \text{ XOR delay.}$$

This is because our examples utilize LSI 100K: 1μ CMOS ASIC technology [30].

It is not difficult to generalize this method for the use of a general p:q compressor rather than a FA. This assumes that such a compressor can be built more efficiently than the one built from the FAs and that delay dependencies are known for the given technology of implementation.

An example representing the ninth VCS of a 12 × 12 multiplier is shown in Fig. 8. The numbers represent the delays that are associated to the nodes. The zero delays are the partial products and the nonzero delays at the top are carry signals originating from the previous slice. Since the number of inputs is even, the first cell used by the program is a HA. The reduction is achieved in five XOR delays.

In our example, we choose to normalize all the signal delays with respect to XOR gate delay and express delays as fractions of it. When different technology is used or the circuits are designed differently than in our case, the ratio of these delays can be quite different. For example, in [31] the sum delay is 1.5 times longer than the delay of the carry bit. However, even in such a case, re-connecting *fast-inputs* and *fast-outputs* according to the method presented here may result in the speed improvement of up to 18%.

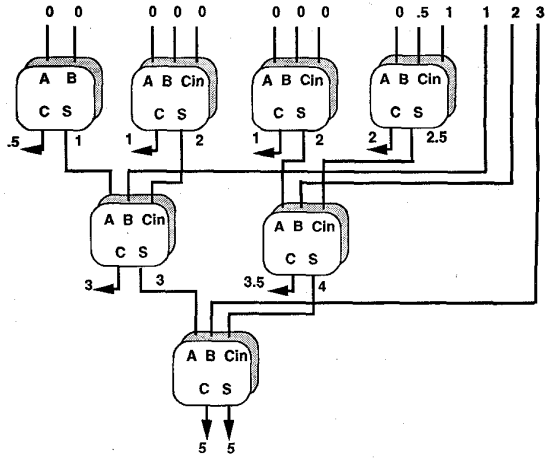


Fig. 8. The ninth VCS of a multiplier.

Although no Full Adder arrangement was assumed, the method produces a tree structure. This result was predictable since the tree structure is theoretically the fastest structure. As illustrated in the Fig. 8, the signals that have the longest delays will actually skip the next level of Full Adders. When it becomes impossible to skip more levels, they are connected to C_{in}, and when all C_{in} are occupied, they are connected to A or B. Since the delay of the carries are also calculated and used, the scheme minimizes both vertical and horizontal critical paths.

Table 1 compares the critical path in the partial product array produced by TDM and of the other schemes. The length of critical paths is estimated in the number of XOR levels. The delay of the *critical path* is dependent not only on the technology of implementation but also on the circuit family (dynamic or static) as well as layout and wiring delays. Therefore, it is difficult to give exact comparison, and the ones given in Table 1 are only reasonable estimates of the relative differences in delays with respect to other schemes [7], [11], [16]. We have decided to normalize all the delays to that of the XOR gate and use the XOR gate delay to represent respective delays in the multiplier tree structure. This decision is justified by the observation that the *critical path* in the multiplier tree indeed consists of a path through a series of XOR gates and that the ultimate speed of the multiplier (Final Adder included) indeed depends on how fast XOR gate can be implemented in a particular technology. A good example supporting this decision can be found in the pass-transistor multiplier implementation by Okhubo et al. [33].

This however, does not affect our method which is based on the *real delays* as calculated for the particular *counter* used in the multiplier under consideration.

TABLE 1
COMPARISON BETWEEN TDM AND OTHER REPRESENTATIVE SCHEMES, IN XOR LEVELS USED IN THE PARTIAL PRODUCT ARRAY

Multiplier Word-length	Wallace Tree [7]	4:2 Tree [11]*	Fadavi-Ardekani [16]	TDM
3	2	2	2	2
4	4	3	3	3
6	6	6 (5)	5	5
8	8	6	7	5
9	8	8	7	6
11	10	9 (8)	8	7
12	10	9 (8)	8	7
16	12	9	10	8
19	12	12 (11)	11	9
24	14	12 (11)	12	10
32	16	12	13	11
42	16	15 (14)	14	12
53	18	15	15	13
64	20	15	16	14
95	20	18 (17)	17	15

* *Number in parenthesis represent delays when a Full Adder is used (instead of 4:2 compressor) every time the column size is found to be three.*

Our algorithm and method can be extended further by going one level deeper into the hierarchy of design. We can treat the multiplier tree as a collection of *interconnected gates* or even *transistors* (as done in [33]) with designated *fast* and *slow* inputs and outputs. We leave this problem for the future extensions of this work suggesting this as a direction for possible further speed optimizations. However, assuming that we have optimized the signal paths using all the available circuit techniques in a particular counter, our method should yield the same results. Wiring delays were not given full consideration in our method.

The average wire delay is rather included into the gate delay as a function of fan-out. If the method presented here is to be fully integrated into the silicon compiler system, wire delay can be included and delays recalculated in an iterative process as each level of the bit reduction matrix is produced. In such a case it might be necessary to re-run the algorithm several time for possible corrections of the interconnection pattern and for fine-tuning of the delay list.

In the Wallace tree implementation, the number of XOR levels is simply the number of Full Adder levels multiplied by two. The improvement using TDM is up to 30%.

3.6 Hardware Complexity

In the following section, we show that the number of cells used is the same as the number produced by Dadda optimization.

The number of inputs to VCS_i is the sum of the number of partial products, called P_i, and the number of carries out from VCS_{i-1} which is actually $C_{i-1} - 1$, since every cell of VCS_{i-1} produces a carry out except the one that is connected to the CPA. We have already shown that: $CN_i = \lfloor I_i/2 \rfloor$

THEOREM 1. *For $i \in [3, N-1]$ the number of cells in VCS_i, $CN_i = CN_{i-1} + 1$.*

PROOF.

The proof is by mathematical induction.

1) Initial step: It is true for I = 3. In this case, $CN_3 = 2$ and $CN_2 = 1$. Therefore, the theorem is true for the initial step
2) Induction hypothesis: $CN_i = CN_{i-1} + 1$.
3) Using the relation $CN_{i+1} = \lfloor (P_{i+1} + CN_i - 1/2 \rfloor$ and, since the induction hypothesis is $CN_i = CN_{i-1} + 1$ and $P_{i+1} = P_i + 1$ for $i \in [3, N-1]$, it follows that:

$$CN_{i+1} = \lfloor (P_i + CN_{i-1} - 1)/2 + 1 \rfloor = \lfloor (P_i + CN_{i-1} - 1)/2 \rfloor + 1.$$

Hence, $CN_{i+1} = CN_i + 1$ for $i \in [3, N-1]$.

THEOREM 2. *For $i \in [N, 2N-2]$ the number of cells in VCS_i, $CN_i = CN_{i-1} - 1$.*

PROOF.

1) Initial step: $CN_{N-1} = N - 2$ (application of the Theorem 1). For i = N,
$CN_N = \lfloor (P_N + CN_{N-1} - 1)/2 \rfloor = \lfloor (2N-5)/2 \rfloor = N - 2.$
Then, for i = N + 1
$CN_{N+1} = \lfloor (P_{N+1} + CN_N - 1)/2 \rfloor = \lfloor (2N-6)/2 \rfloor = N - 3.$
Therefore, the theorem is verified for the initial step
2) Induction hypothesis: $CN_i = CN_{i-1} - 1$.
3) Using the relation $CN_{i+1} = \lfloor (P_{i+1} + CN_i - 1) / 2 \rfloor$ and, since the induction hypothesis is $CN_i = CN_{i-1} - 1$ and $P_{i+1} = P_i - 1$ if $i \in [N, 2N-2]$, we find:

$$CN_{i+1} = \lfloor (P_i + CN_{i-1} - 1)/2 - 1 \rfloor = \lfloor (P_i + CN_{i-1} - 1)/2 \rfloor - 1.$$

Hence, $CN_{i+1} = CN_i - 1$ for $i \in [N, 2N-2]$.

THEOREM 3. *The total number of cells in the partial product array is $(N-1)(N-2)$*

PROOF.

VCS_0 and VCS_1 have no cells. The numbers of cells for VCS_2 to VCS_{N-1} represent an arithmetic progression $1 + 2 + 3 + \ldots + N - 2$. That is:

$$\sum_{i=1}^{N-2} i = \frac{(N-1)(N-2)}{2}.$$

The number of cells in VCS_N to VCS_{2N-2} is similarly shown to be $(N-1)(N-2)/2$ by using arithmetic progression $N - 2 + N - 1 + \ldots + 1$.

Therefore, the total number of cells is:

$$\sum_{i=0}^{2N-2} i = \sum_{i=0}^{N-1} i + \sum_{i=N}^{2N-2} i$$

$$= \frac{(N-1)(N-2)}{2} + \frac{(N-1)(N-2)}{2} = (N-1)(N-2).$$

The number of FA and HA produced by our program is exactly $(N-1)(N-2)$. Although Dadda did not give any formula [8], the results of his optimization are 110 cells for 12-bit multiplication and 506 cells for 24-bit multiplication. Those numbers correspond to our formula $(N-1)(N-2)$. Therefore, we claim that our method is also optimized in terms of number of cells. Fadavi-Ardekani (F-A) scheme produces 121 and 582 cells, respectively, for 12- and 24-bit multiplier sizes.

3.7 Comparison and Experimental Results

We have designed 4:2, 9:2, F-A, and TDM partial product arrays in 1μ CMOS-ASIC technology for a 24-bit multiplier. The results that were obtained by simulation using timing simulator from LSI Logic and LSI 100K [30] timing information under nominal conditions [T = 25°C, V_{cc} = 5V]. The results for the critical path delay in the partial product array are summarized in the Table 2. LSI Logic simulator takes into account loading due to different fan-outs and includes an estimated wire load. This estimation is based on the fan-out of the particular output node. Our past experience with the simulator based on comparisons with actual fabrications has been very good. However, our results are limited to simulation and they do not represent actual measurements. It is also appropriate to mention that our results do not reflect the complexity of wiring. The wiring required for a multiplier produced using the TDM method is not substantially different from other schemes such as Wallace or Dadda, knowing that the number of counters used by our method is comparable to Dadda's [8] and that the main difference is in the way inputs and outputs of the counters are connected locally, not in the additional connections. However, using an improved Dadda's scheme a multiplier can be designed with only local interconnections as done in [32]. TDM scheme uses more complex wiring than [32] and an *array multiplier* such as [31]. The use of (4, 2) compressors results in less complex wiring and layout.

TABLE 2
CRITICAL PATH DELAY [CMOS: LEFF = 1μ, T = 25°C, V_{CC} = 5V]

N = 24 bits	4:2 Design	9:2 Design	Fadavi-Ardekani	TDM Design
Delay [nS]	14.0	13.0	11.7	10.5

Fig. 9 depicts a comparison in terms of XOR levels between a regular Wallace/Dadda scheme, an optimized 4:2, the Fadavi-Ardekani scheme and the TDM scheme. The delay for the partial product array includes partial product generator delay consisting of a simple row of AND gates. In the next section, we consider the delay component of an optimal CPA.

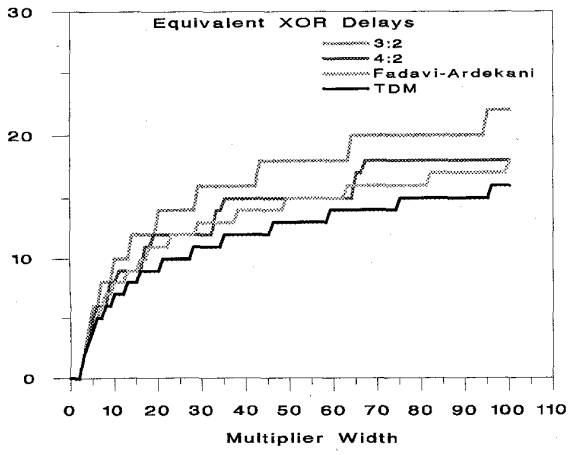


Fig. 9. Comparison of (3, 2), 4:2, F-A, and TDM schemes.

For this particular 24-bit CMOS implementation, the TDM Scheme is 33% faster than 4:2 and 25% and faster than 9:2. The improvement over F-A method is 11% in terms of delay and 15% in terms of the number of cells used.

4 Speed Improvement in the Final Adder

Finally, multiplier speed can be further improved via optimization of the CPA to the nonuniform signal arrival profile of its inputs. It is well-known that the signals applied to the inputs of the CPA arrive first at the ends of the CPA and the last ones are the signals fed to the bits in the middle of the CPA [26]. The shapes of the signal arrival profile originating from the CPA and the multiplier tree of an 13 × 13-bit ASIC multiplier [19] are shown in Fig. 10.

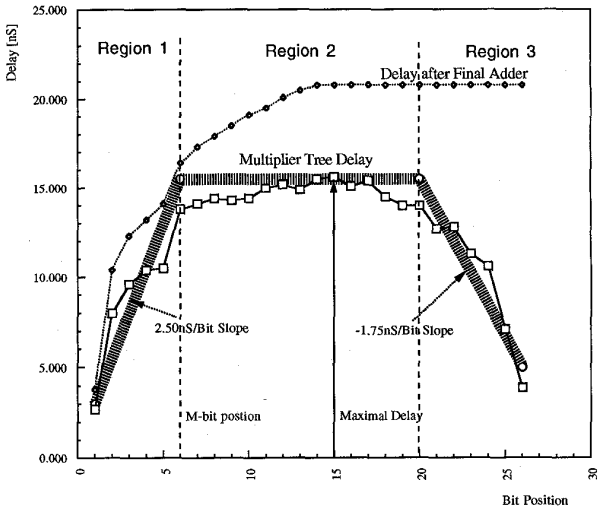


Fig. 10. Signal Arrival Profile and selection of the adder types in the three regions of the multiplier.

4.1 The Choice of Final Adder

All the known schemes for fast addition are developed under the assumption of an uniform signal arrival profile. The first problem is in selecting one of the CPA schemes that are most adequate to be used in the multiplier. It is obvious that we should use the fastest scheme since final addition time is a significant addendum to the critical path of the multiplier.

There are three regions to be considered with respect to the worst case signal arrival profile from the multiplier tree as shown in Fig. 10. Region 1 has a positive slope with respect to bit position. To use an adder which adds faster than this slope would not make much sense and would be a waste of hardware resources. The type of adder used in Region 1 is determined by this slope. This slope is determined by the fact that the arrival of bits is incrementally delayed by a path traversing a FA used in bit compression. Therefore, using any of the more powerful and, therefore, more complex, schemes for this part of the multiplier is not justified. If ripple-carry adder speed can not match this

slope, good choices are simple and VLSI efficient schemes such as Variable Block Adder (VBA) [23]. A simple analysis shows that CLA has worse performance in this region [27].

From the point of the maximal delay (M) which is usually in the middle bit position (or skewed a few bits toward the most significant side) the addition has to be performed in the fastest possible way. Therefore, in Regions 2 and 3, addition has to be very fast because the addition time Δ_{ADD_2} is a direct addendum to the multiplier delay. Analysis shows that the best choices for the adder in this region are: Conditional Sum Adder (CSA) and the Carry Select Adder (CSLA) [21], [27]. Choice of CSLA scheme results in less complex implementation simply because CSLA is a subset of CSA. The difference in speed between CSA and CSLA diminishes as the input arrival profile is changed from uniform to nonuniform [21]. Though the CSLA adder is still slower than the CSA adder, this difference is offset by the relative simplicity of its implementation which is preferred by most designers. Smaller size and simpler layout of CSLA would further affect the relative speed difference, reducing the advantage of CSA. Given that fact, Carry Select Addition (CSLA) is the best choice for Region 2 [22].

The adder constructed for this part (Regions 2 and 3) can be further subdivided into two. Region 2 requires the fastest addition available, the choice of which is Carry-Lookahead (CLA) adder, or optimized derivatives of CLA [24] combined into a CSLA, all of which depend on the particular circuit and technology used.

In the region of the negative slope (Region 3), where the most significant bits arrive first, the most suitable choice is again CSLA. However, CSLA adds time for selection multiplexers, ΔMUX, which is not negligible given that the adder sections are already constructed to be in the same speed range. It is also obvious that due to the steep negative slope in this region, there is substantial time left to add the bits in the most significant position before a selection process occurs. Simple analysis shows that for this part, VBA is a much better candidate than CLA or for that matter any other scheme [27].

Determination of bit positions separating the Regions 1, 2, and 3 is based on intersecting the bit arrival time from the adders used in the corresponding regions, so that the selection in CSLA is done at the appropriate time. This is a relatively complex iterative process which is illustrated in Fig. 11. As can be inferred from Fig. 11, this is an iterative technique which does not require a large number of iterations, given that the bit positions S_1 and S_2 are integers. The total delay of the multiplier is:

$$\Delta_{MULT} = \Delta_{TREE} + \Delta_{ADD_2} + \Delta_{MUX}$$

The resulting CPA structure is shown in Fig. 12a. The signal arrival profiles originating from the tree of an 13 × 13-bit multiplier for two different input patterns applied are shown in Fig. 12b. It is interesting to observe that while the pattern B results in a faster signal arrival from the multiplier array than pattern A, the product originating from the CPA for the pattern B has a longer delay than the pattern A. This example illustrates the point that it is more important to *tune* the CPA into the signal arrival profile than to apply the fastest available addition scheme. Further analysis of CPA optimization can be found in [27].

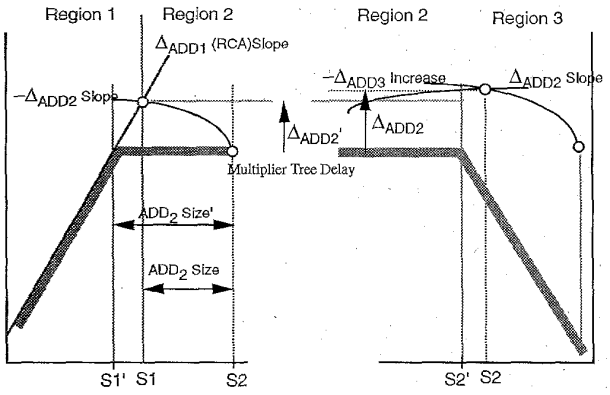


Fig. 11. Determination of bit positions S_1 and S_2 determining the size of the adders used: Left—Structure of the Final Adder; Right—Signal Arrival Profile.

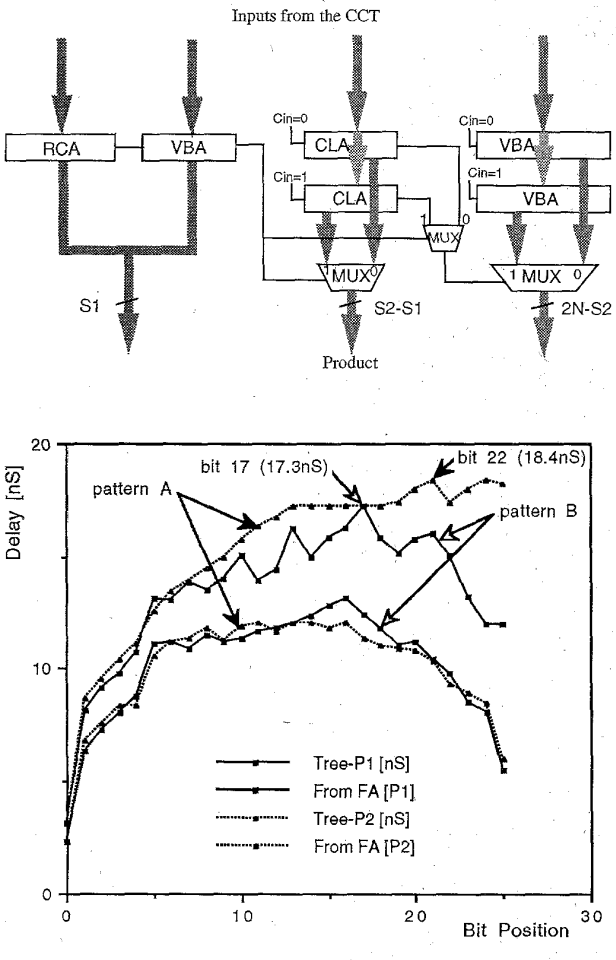


Fig. 12. Structure of the Final Adder (top); Signal Arrival Profile from the column compression tree and the Final Adder (bottom).

5 Conclusion

The presented algorithm and method for parallel multiplier implementation takes advantage of the uneven delays through a Full Adder in order to build a global compressor that minimizes the critical path of the multiplier. The compression tree is divided into vertical slices that are optimized globally to produce individual Vertical Compression Slices (VCS). This minimization does not only involve the vertical signal path, but also involves the horizontal signals from the previous VCS. A method to implement a speed optimized multiplier tree which includes an algorithm for net-list generation has been presented. This method has been implemented using C language, although it would not be difficult to describe in a hardware description language such as VHDL, or to implement the algorithm as a part of a silicon compiler. A multiplier produced by this algorithm has been implemented in 1μ CMOS technology together with several competing schemes. The results obtained using our algorithm are better not only in terms of speed but surprisingly also in terms of the number of cells used. The use of a compressor optimized at the transistor level instead of a Full Adder could result in further improvements of speed and can be easily incorporated in the presented method.

Acknowledgments

The authors are grateful to the reviewers. We are also grateful to Prof. Charles Martel for helping us understand the complexity of the algorithm. We express our gratitude to LSI Logic and Cascade Design Automation for supporting us generously with their tools, and to Antonio de la Serna for his careful reading of our text.

This research was supported by California Micro Grant No. 93-118 and by a grant from the Office of Research at the University of California at Davis.

References

[1] E.E. Swartzlander, *Computer Arithmetic*, vols. 1 and 2, IEEE CS Press, 1990.

[2] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*. John Wiley and Sons, 1979.

[3] S.D. Pezaris, "A 40ns 17-bit Array Multiplier," *IEEE Trans. Computers*, vol. 20, no. 4, pp. 442-447, Apr. 1971.

[4] A.D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanical Applications in Math.*, vol. 4, part 2, pp. 236-240, 1951.

[5] O.L. MacSorley, "High Speed Arithmetic in Binary Computers," *IRE Proc.*, vol. 49, pp. 67-91, Jan. 1961.

[6] D. Villeger and V.G. Oklobdzija, "Evaluation of Booth Encoding Techniques for Parallel Multiplier Implementation," *Electronics Letters*, vol. 29, no. 23, Nov. 1993.

[7] C.S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Computers*, vol. 13, no. 2, pp. 14-17, Feb. 1964.

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

[9] W.J. Stenzel, "A Compact High Speed Parallel Multiplication Scheme," *IEEE Trans. Computers*, vol. 26, no. 2, pp. 948-957, Feb. 1977.

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

[11] M.R. Santoro, "Design and Clocking of VLSI Multipliers," PhD dissertation, Technical Report No. CSL-TR-89-397, Oct. 1989.

[12] M.R. Santoro, "A Pipelined 64x64b Iterative Array Multiplier," *Digest of Technical Papers, Int'l Solid-State Circuits Conf.*, Feb. 1988.

[13] M. Nagamatsu et al., "A 15nS 32x32-bit CMOS Multiplier with an Improved Parallel Structure," *Digest of Technical Papers, IEEE Custom Integrated Circuits Conf.*, 1989.

[14] J. Mori et al., "A 10nS 54x54-b Parallel Structured Full Array Multiplier with 0.5-u CMOS Technology," *IEEE J. Solid State Circuits*, vol. 26, no. 4, Apr. 1991.

[15] P. Song and G. De Michelli, "Circuit and Architecture Trade-Offs for High Speed Multiplication," *IEEE J. Solid State Circuits*, vol. 26, no. 9, Sept. 1991.

[16] J. Fadavi-Ardekani, "M x N Booth Encoded Multiplier Generator Using Optimized Wallace Trees," *IEEE Trans. VLSI Systems*, vol. 1, no.2, June 1993.

[17] G. Bewick, "High Speed Multiplication," *Proc. Electronic Research Laboratory Seminar*, Stanford Univ., Mar. 12, 1993 (also private communications).

[18] V.G. Oklobdzija and D. Villeger, "Multiplier Design Utilizing Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," *Proc. 10th Anniversary Symp. VLSI Circuits*, Taipei, Taiwan, May 1993.

[19] T. Soulas, D. Villeger, and V.G. Oklobdzija, "An ASIC Multiplier for Complex Numbers," *Proc. EURO-ASIC 93, The European Event in ASIC Design*, Paris, France, Feb. 22-25, 1993.

[20] D. Villeger, "Fast Parallel Multipliers," *Final Report, Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique*, Noisy-le-Grand, France, May 11, 1993.

[21] A.K.W. Yeung and R.K. Yu, "A Self-Timed Multiplier with Optimized Final Adder," *Final Report for CS 292I* (Prof. Oklobdzija), Univ. of California at Berkeley, Fall 1989.

[22] O.J. Bedrij, "Carry-Select Adder," *IRE Trans. Electronic Computers*, June 1962.

[23] V.G. Oklobdzija and E.R. Barnes, "On Implementing Addition in VLSI Technology," *IEEE J. Parallel Processing and Distributed Computing*, no. 5, 1988.

[24] B.D. Lee and V.G. Oklobdzija, "Delay Optimization of Carry-Lookahead Adder Structure," *J. VLSI Signal Processing*, vol. 3, no. 4, Nov. 1991.

[25] M. Suzuki et al., "A 1.5nS 32b CMOS ALU in Double Pass-Transistor Logic," *Digest of Technical Papers, 1993 IEEE Solid-State Circuits Conf.*, San Francisco, Feb. 24-26, 1993.

[26] K. Fai-Pang et al., "Generation of High-Speed CMOS Multiplier-Accumulators," *Proc. ICCD-88, Int'l Conf. Computer Design*, Rye, N.Y., Oct. 1988.

[27] V.G. Oklobdzija, "Design and Analysis of fast Carry-Propagate Adder under Non-Equal Input Signal Arrival Profile," *Proc. 28th Asilomar Conf. Signals, Systems, and Computers*, Oct. 30-Nov. 2, 1994.

[28] K.J. Singh et al., "Timing Optimization of Combinational Logic," *Proc. ICCAD 88, Int'l Conf. CAD*, Nov. 1988.

[29] H.J. Touati et al., "Delay Optimization of Combinational Logic Circuits by Clustering," *Proc. ICCAD 91, Int'l Conf. CAD*, Oct. 1991.

[30] *1.0-Micron Array-Based Products Databook*, LSI Logic Corp., Sept. 1991.

[31] J.Y. Lee et al., "A High-Speed High-Density Silicon 8X8-bit Parallel Multiplier," *IEEE J. Solid State Circuits*, vol. 22, no. 1, Feb. 1987.

[32] Z. Wang, G.A. Julien, and W.C. Miller, "Column Compression Multipliers for Signal Processing Applications," *VLSI Signal Processing*. New York: IEEE Press, 1992.

[33] N. Okhubo et al., "A 4.4nS CMOS 54x54-b Multiplier Using Pass-Transistor Multiplexer," *Proc. Custom Integrated Circuits Conf.*, San Diego, Calif., May 1-4, 1994.

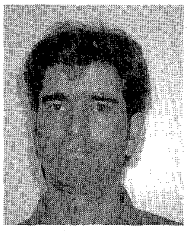
[34] C. Martel, V.G. Oklobdzija, R. Ravi, and P. Stelling, "Design Strategies for Optimal Multiplier Circuits," *Proc. 12th IEEE Symp. Computer Arithmetic*, Bath, England, July 19-21, 1995.

Vojin G. Oklobdzija obtained the Dipl. Ing. (MScEE) degree in electronics and telecommunications from the Electrical Engineering Department of the University of Belgrade, Yugoslavia, in 1971, and was on the faculty there until 1976 when he became a Fullbright Scholar at the University of California at Los Angeles. He obtained his MSc and PhD in computer science from UCLA in 1978 and 1882, respectively.

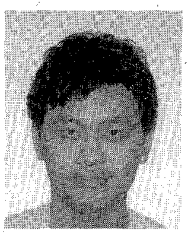
As a visiting faculty member from IBM, Prof. Oklobdzija taught courses in computer architecture, computer arithmetic, and computer design at the University of California at Berkeley from 1988 to 1990. He spent eight years as a research staff member at the IBM T.J. Watson Research Center in New York, where he made contributions to development of RISC and super-scalar RISC architecture and processors. He left IBM in 1991 and today is with Integration, Berkeley, California, and the Electrical and Computer Engineering Department of the University of California at Davis. He has also worked on massively parallel machines and was one of the initiators of the supercomputer project at IBM. His industrial experience includes positions at the Microelectronics Center of the Xerox Corporation, and consulting positions at Sun Microsystems Laboratories, AT&T Bell Laboratories, and various others.

Dr. Oklobdzija has published more than 70 papers in the areas of circuits and technology, computer arithmetic, and computer architecture. He has presented many invited talks in the U.S., Europe, Latin America, Australia, and Japan. He holds one of the patents on the IBMRS/6000—"PowerPC" architecture, and four U.S. and four European patents in the area of circuits and computer design. His current research interests are in VLSI and fast circuits, efficient implementations of algorithms, and computation. His work on the fast ALU scheme has been widely referenced and used.

He is a fellow of the IEEE and serves on the editorial boards of the *Journal of VLSI Signal Processing* and *IEEE Transactions on VLSI Systems.*



David Villeger studied electronics at Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique in Paris, France. He chose to specialize in microelectronics and received the Diplome d'Ingenieur in 1993 after completing his thesis project at the University of California at Davis. He spent time in Japan, giving presentations at the Yokogawa Denki Corporation in Tokyo and at Osaka University. His current research interests are in computer arithmetic and VLSI. He has worked on square root and multiplication algorithms, and has published four journal and four conference papers on those subjects. His consulting experience involves numerous companies such as Sun Microsystems and Hewlett-Packard. Currently, he in working as a consultant to Silicon Graphics Corporation in Mountain View, California.



Simon S. Liu received the BS degree in electrical engineering in 1991 and the MS degree in 1994, both from the University of California at Davis. The work he did that is related to this paper was done while he was at UC Davis. During the spring and fall of 1992, he was with Advanced Micro Devices, Sunnyvale, California, as a co-op, and worked on improving cell characterization and time verification CAD tools. In 1993, he joined AMD, where he had been engaged in the research and development of VLSI CAD. He is currently working as a design engineer and is involved in the PCI bus interface project.