Multiplier Reduction Tree with Logarithmic Logic Depth and Regular Connectivity

H. Eriksson†, P. Larsson-Edefors†, M. Sheeran‡, M. Själander†, D. Johansson†, and M. Schölin†
†VLSI Research Group, ‡Formal Methods Group
Department of Computer Science and Engineering
Chalmers University of Technology, SE-412 96 Göteborg, Sweden
{hener,perla}@ce.chalmers.se, ms@cs.chalmers.se

***Abstract*— A novel partial-product reduction circuit for use in integer multiplication is presented. The High-Performance Multiplier (HPM) reduction tree has the ease of layout of a simple carry-save reduction array, but is in fact a high-speed low-power Dadda-style tree having a worst-case delay which depends on the logarithm ($\mathcal{O}(\log N)$) of the word length N.**

I. Introduction

The integer multiplier is a very important and wide-spread building block of digital designs. When multiplier delay is a critical design parameter, designers tend to use well-known logarithmic multipliers such as Wallace [1], Dadda [2], or TDM [3]. This class of multipliers is based on a *reduction tree*, in which different schemes of compression of partial-product bits can be implemented, all with the result that the worst-case multiplier delay depends logarithmically on the factor word length.

To date, no regular way of inter-connecting the adding cells in such reduction trees has been proposed[1] and, thus, textbook array multipliers [5] are seriously considered [6] if layout regularity is the main concern and speed is not. It has, however, been shown by Callaway and Swartzlander [7] that not only speed is sacrificed when an array multiplier is used instead of a logarithmic one, but also *power*. When comparing netlists of the reduction trees of a 32-b Wallace, a 32-b Dadda, and a 32-b TDM multiplier with the partial-product compression array of a 32-b carry-save multiplier, the reduction trees on the average proved to be 2.3× faster *and* 3.0× more power efficient than the array [8].

We propose a new organization of the reduction tree, which is based on a partial-product compression similar to the Dadda approach. The connectivity of the adding cells in the triangle-shaped *High-Performance Multiplier (HPM)* reduction tree is completely regular. At the same time it exhibits a worst-case delay which depends on the logarithm ($\mathcal{O}(\log N)$) of the word length N.

II. The HPM Reduction Tree

Typically parallel multipliers comprise three parts: *primary partial-product bit generation*, performed either by simple AND gates or Booth [9] or modified Booth recoding strategies [10]; *partial-product bit compression*, using either an irregular logarithmic tree or a regular array; and *final addition*, consuming the output of the partial-product bit compression with an adder whose performance is tailored to match the output delay profile of the compression circuitry [3].

The focus of this paper is on a reduction tree that is easy to place and route *and* has a logarithmic logic depth. We will describe a connectivity strategy that enables a regular organization of the cells in the reduction tree. However, first we need to select a multiplier to start from, our *model* multiplier. In principle, we can select any of the three logarithmic multipliers that were mentioned in the introduction: Wallace, Dadda, or TDM multipliers. Although it is claimed [11] to be less suitable for custom implementations than Wallace, we have selected the Dadda multiplier as our model multiplier. The main reason for this choice is that regardless of the word length N the Dadda reduction tree always makes use of a minimal amount of circuitry, M_{OPT} [3]:

$$M_{OPT} = (N-1) \cdot (N-2) \tag{1}$$

Here, $N-1$ cells are half adders, and the remainder are full adders.

Regarding the connectivity of cells, we are assuming a triangular reduction tree rather than a rectangular. The reason for this choice is that a triangular cell placement has a shorter total wire length than a rectangular one (see Sec. II-A). Note that the unused area within a bounding box surrounding the triangular reduction tree could be used for partial-product bit generating logic and decoupling capacitance. Thus the resulting circuit footprint is rectangular also in this case.

A. Design Method

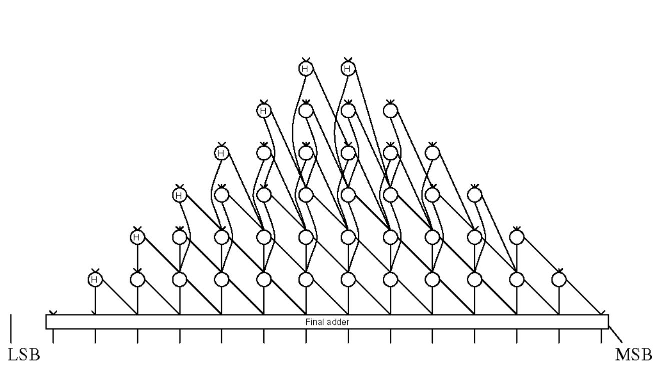
The Dadda is closely related to the Wallace multiplier in that they both have an $\mathcal{O}(\log N)$ reduction in the reduction tree. In comparison to the Dadda multiplier algorithm, the Wallace algorithm creates a shorter final pair of rows. There are claims [11], [12] that the shorter final adder makes the Wallace multiplier faster than Dadda. This is not true, since it is the delay profile from the reduction tree that matters and the delay profiles are almost identical for Wallace and Dadda multipliers.

The four steps needed to obtain an 8-b Dadda multiplier are shown in Fig. 1. The 64 partial-product bits are compressed into two rows, which are fed to a fast final adder. In this representation, where carry, sum, and *primary* partial-product bits are all represented as dots, it is hard to interconnect the adding cells in a regular fashion. However, if special care is taken during the assignment of adding cells, the result is the logarithmic depth High-Performance Multiplier (HPM) reduction tree shown in Fig. 3(a). Circuitry for primary partial-product bit generation is omitted in the figure—the very short wires represent primary partial-product bits.

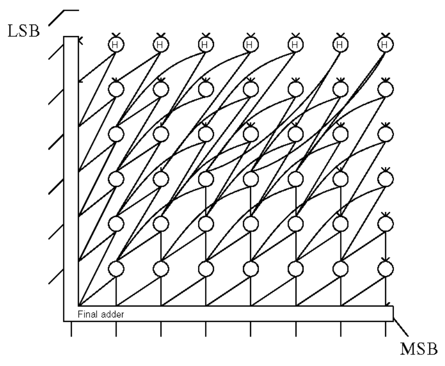
The three-over-two compression is visible when inspecting Fig. 3(a) since every other row of adding cells takes two carry vectors and one sum vector as inputs, whereas the other rows take one carry vector and two sum vectors. The encircling scheme leading to an HPM tree is shown in Fig. 2.

Fig. 3 illustrates why a triangular cell placement is preferred for the HPM: wire routing is simple and thus it is easy to write a generator—or to make a custom design. Also, if the wires used to route the

[1]Luk and Vuillemin [4] introduced a regular layout of a reduction tree with logarithmic depth, however, since the primary outputs of the tree are embedded in the center of the layout, it is difficult to interface the reduction tree with the final adder.



(a) Triangular



(b) Rectangular

Fig. 3. Different cell placements for the HPM reduction tree. The rectangular shape has larger total wire length.

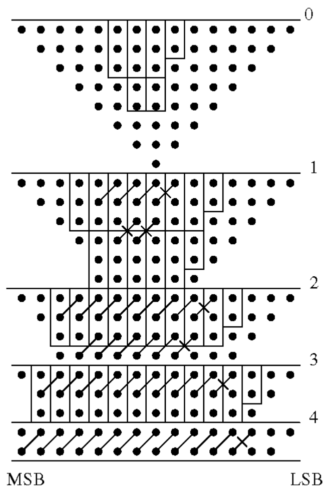


Fig. 1. The four steps leading to an 8-b Dadda multiplier. Two dots that are tied together by a line represent a sum and a carry. If the line is crossed, it is a half adder producing the sum and carry, otherwise it is a full adder. The cell type is also indicated by the number of bits that are encircled.

primary partial-product bits are ignored, the total wire length for an 8-b multiplier is approximately 38% shorter for the triangle compared to the rectangle. (Here, we assumed square-shaped adding cells and that 45 degree wires are allowed. The estimated wire length is calculated as the distance between the cells it connects.) The primary partial-product bits can be generated locally within the reduction tree, or outside it; either way more wires than just sum and carry wires need to be routed inside the reduction tree.

Modified Booth recoding could, of course, be employed to reduce the number of primary partial-product bits, but that is out of the scope of this paper. However, there is no difference between the HPM tree and a Wallace (or a Dadda) tree in the way Booth recoding is used.

B. Layout Organization

In the following, we assume that the primary partial-product bits are generated outside the tree. We have implemented both the HPM reduction tree, and for comparison, the reduction circuitry from the carry-save array (CSA) multiplier, which is well known for its regular connectivity. The CSA multiplier was slightly modified to better fit

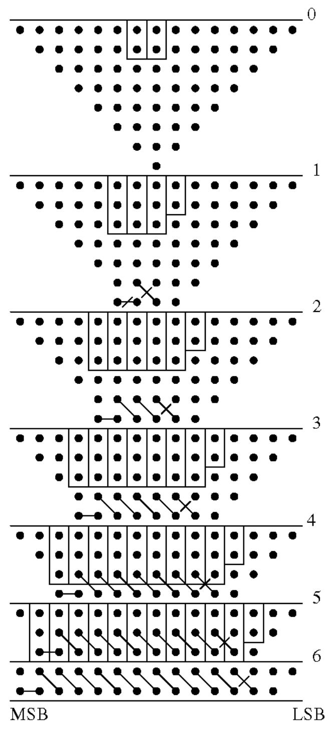


Fig. 2. The six steps leading to an 8-b HPM tree. Note that the six steps do *not* imply a linear logic depth. The logic depth is still logarithmic.

the comparison with the HPM reduction tree: the carry-save concept was used in a triangular minimal-hardware placement, rather than in the more common rectangular placement.

We have found a way to build the reduction circuitry for *both* the CSA and HPM multipliers using the *same overall structure of cells*, which is depicted in Fig. 4. Our task now is to design the individual cells so as to obtain the desired routing between cells.

We start with the cells containing half adders. Figs 5(a) and 5(d) show the over-the-cell routing patterns for the HPM and CSA respectively. K partial-product bits enter at the top, and the role of the HA cell is to reduce their number by one and to produce an output carry that is passed rightwards to the next column. A half adder is

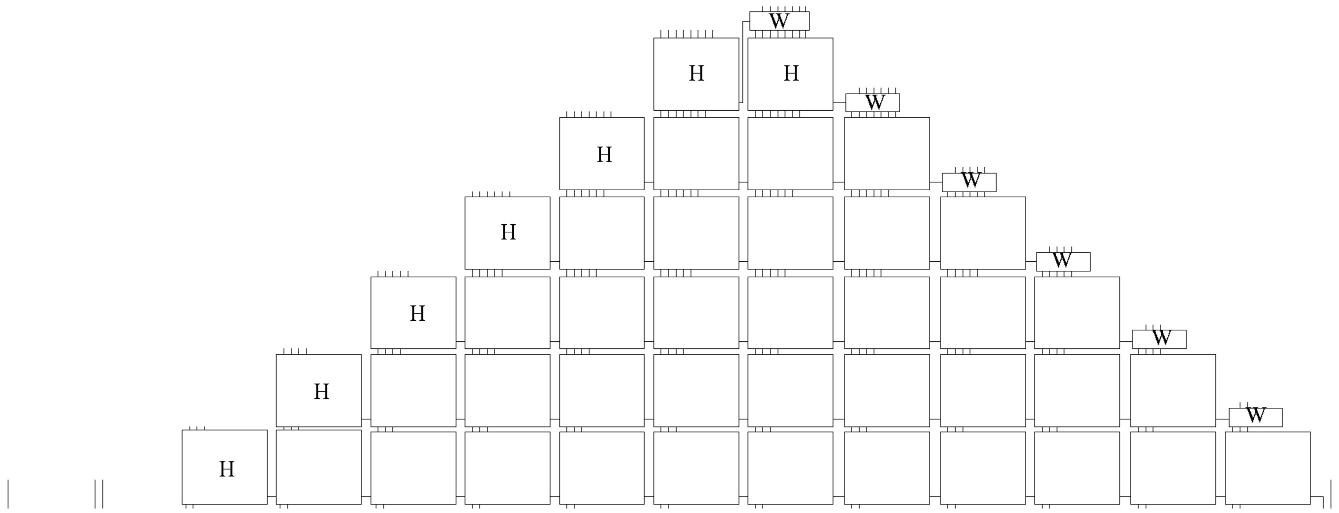


Fig. 4. The overall reduction structure used to build both the HPM and CSA multipliers. Boxes marked H contain half adders and wiring, unmarked boxes contain full adders and wiring, and cells marked W contain only wiring.

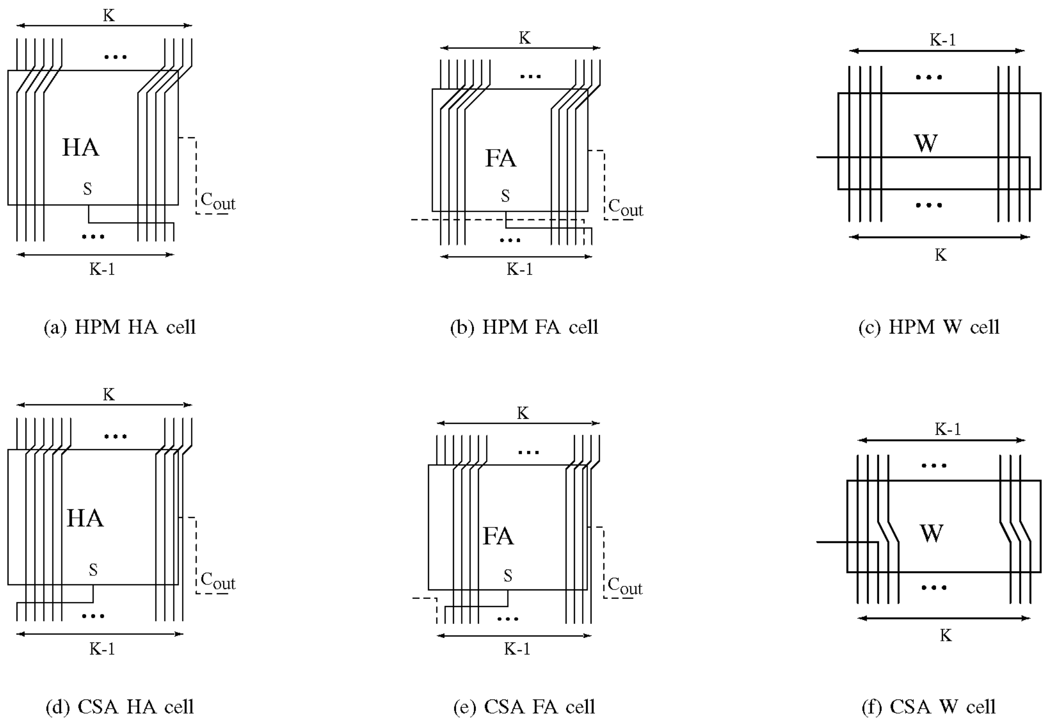


Fig. 5. The routing patterns for full adder, half adder and wiring cells in the HPM and the CSA multipliers. In both cases, these cells are combined into the structure shown in Fig. 4. As long as primary partial-product bits remain to be compressed, the HPM compresses only those, ignoring sum and carry bits as long as possible. This is in contrast to the CSA, in which most FA cells consume one primary partial-product bit for each compression stage.

used to convert the leftmost two partial-product bits into a sum and an output carry. The output carry is passed rightwards, and the only remaining question is how to place the sum bit among the remaining $K-2$ partial-product bits, to give the desired $K-1$ outputs. For the HPM, we have already decided to process sum and carry bits as late as possible, so we place the sum bit as far to the right as possible, since the adding cells process bits from the left. For the CSA, the sum bit should be processed by the cell below this one, so we place the sum bit as far to the left as possible. The cells containing full adders are a little more complicated, because they also have an input carry bit. However, the strategy is similar: For the HPM, we process the sum bit as late as possible, as before, and we also process the input carry as late as possible, Fig. 5(b). Thus, these two bits are passed as far to the right as possible. For the CSA, these two bits are instead passed as far to the left as possible, so that they are processed by the cell immediately below this one, Fig. 5(e). Similarly, the W cells are designed to give the required connectivity, Figs 5(c) and 5(f).

As a sanity check, consider the leftmost of the two tallest columns in the overall structure shown in Fig. 4. This column receives 8 primary partial-product bits from the top, and 5 input carry bits from the left. The 6 cells produce a total of 6 sum bits. The following sequence indicates the order in which these 19 bits are consumed: [pp, ppp, ppp, scs, csc, scs, cs], starting with the two primary partial-product bits consumed by the half adder, and ending with the carry/sum pair that is the output at the bottom of the column. The remaining groups of three are the inputs to successive full adders as we move down the column. Here, the pattern of alternating sum and carry bits, processed late, is clear. For the same column in the CSA, however, the pattern is [pp, spp, csp, csp, csp, csp, cs]. The input carry and sum for a given cell are consumed in the cell below. The reader might like to compare these patterns with the relevant columns in Fig. 3(a).

III. Evaluation

First, a 16-b HPM reduction tree was formally verified using gate-level VHDL netlists in a functional equivalence-checking tool (Formality). Then we evaluated the HPM tree in three different ways: *i)* To validate the logarithmic delay, we compared HPM to both Wallace and TDM trees. *ii)* To evaluate the relative crosstalk sensitivity of HPM, we compared it to the CSA array. *iii)* Finally, to make sure the glitch power associated with regular reduction circuits is effectively suppressed, we compared the power dissipation of HPM to the CSA array.

A. Timing Comparison (i)

We used PathMill to analyze our 0.35-μm schematic designs of HPM, Wallace (WAL) and TDM. Since we analyzed several wordlengths, we used a generator that inserted DC wire loads that had been calibrated to actual layouts. The delay values are presented in Table I. Note that the delay values are for complete multipliers, when a Kogge-Stone adder [13] is used as the fast final adder. Here identical adding cells were used for all multiplier types, that is, we did not take advantage of the predictable cell placements to further increase HPM multiplier speed by circuit sizing.

TABLE I
Simulated delay values for complete multipliers.

Type	WAL		HPM		TDM	
N	32	54	32	54	32	54
Delay [ns]	8.22	10.6	7.94	10.5	7.80	10.2

All multiplier exhibit delays that are logarithms of wordlength. As expected the TDM is the fastest, but in exchange it requires the largest design effort, since it has irregular connectivity and an elaborate selection/assignment of cells. The WAL multiplier is slightly slower than the HPM and this may be due to its more complex wiring.

B. Timing Comparison (ii)

Due to its regularity, the HPM tree has some critical wires that run in parallel for a relatively long distance. Thus, crosstalk-induced delay may be a serious issue. To uncover potential problems with crosstalk in critical inter-cell wires, we used PathMill to compare the crosstalk sensitivity of a 16-b HPM to that of a 16-b CSA, which has no long wires in parallel, but many shorter. We used netlists of fully extracted 0.13-μm layouts of complete reduction trees. The results are as follows: The HPM had a 43% increase in delay when comparing a netlist *without* crosstalk to one *with* crosstalk capacitances. For the CSA the delay increase was 37%. Thus, the crosstalk of an HPM tree is not significantly worse than other trees. On the contrary, it should be noted that since wiring in the HPM tree is regular, increased wire spacing can be applied in a systematic way to reduce crosstalk.

C. Power Comparison (iii)

We also compared the power dissipation of the HPM reduction tree to the CSA reduction circuitry using HSpice. Here we used post-layout extracted 0.35-μm netlists together with random input data. In our analysis we found that a 16-b HPM reduction tree is 1.9$\times$ more power efficient than the CSA reduction array, and this is due to a smaller number of transitions [7].

IV. Conclusion

We have introduced a new reduction tree for integer multiplication, the High-Performance Multiplier (HPM) reduction tree, which has an $\mathcal{O}(\log N)$ delay dependence on word length N. The connectivity of adding cells in the reduction tree is regular and we have shown that routing becomes trivial, which can be utilized in any type of design method; fully automatic, custom, or somewhere in between. In contrast to other logarithmic multipliers, like Wallace, the design effort for a custom-made HPM multiplier is very limited; in fact we showed that it is as low as for a textbook array multiplier. The predictable wiring resulting from the regularity of the HPM tree can enable both systematic sizing of logic circuitry and systematic wire spacing engineering so as to minimize total multiplier delay.

V. Acknowledgments

Thanks to Per Bjesse at Synopsys, Inc., who verified the HPM multiplier using Formality. We gratefully acknowledge equipment grants from Intel Corporation to each of our research groups.

References

[1] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. 13, pp. 14–17, Feb. 1964.
[2] L. Dadda, "Some Schemes for Parallel Adders," *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, May 1965.
[3] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 294–306, Mar. 1996.
[4] W. K. Luk and J. E. Vuillemin, "Recursive Implementation of Optimal Time VLSI Integer Multipliers," in *Proceedings of VLSI'83*, 1983, pp. 155–168.
[5] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, A Design Perspective*, Prentice Hall, second edition, 2003.
[6] J. T. Kao, M. Miyazaki, and A. P. Chandrakasan, "A 175-mW Multiply-Accumulate Unit Using an Adaptive Supply Voltage and Body Bias Architecture," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1545–1554, Nov. 2002.
[7] T. K. Callaway and Jr. E. E. Swartzlander, "Optimizing Multipliers for WSI," in *Proceedings of the Fifth Annual IEEE International Conference on Wafer Scale Integration*, 1993, pp. 85–94.
[8] H. Eriksson, *Efficient Implementation and Analysis of CMOS Arithmetic Circuits*, Ph.D. Thesis, Chalmers University of Technology, 2003.
[9] A. D. Booth, "A Signed Binary Multiplication Technique," *Quarterly Journal Mechanical and Applied Mathematics*, vol. 4, pp. 236–240, 1951.
[10] W.-C. Yeh and C.-W. Jen, "High-Speed Booth Encoded Parallel Multiplier Design," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 692–701, July 2000.
[11] M. J. S. Smith, *Application-Specific Integrated Circuits*, Addison-Wesley Publishing Company, first edition, 1997.
[12] K. C. Bickerstaff, M. Schulte, and Jr. E. E. Swartzlander, "Reduced Area Multipliers," in *Proceedings of the International Conference on Application-Specific Array Processors*, 1993, pp. 478–489.
[13] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. 22, no. 8, pp. 786–793, Aug. 1973.