A Theoretical Analysis of Square versus Rectangular Component Multipliers in Recursive Multiplication

Behrooz Parhami, *Life Fellow, IEEE*

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106-9560, USA

parhami@ece.ucsb.edu

Final version: November 28, 2016

ABSTRACT: *Previous discussions of recursive multiplication in the literature focus on how/why the scheme works and how a multiplier of a desired size can be built from given component multipliers or multiply-add modules. The form factors (square versus rectangular) for the component multipliers and the one to be synthesized, and how they affect the performance and cost of the resulting multiplier, have not been contemplated. We remedy this shortcoming by dealing with the general problem of synthesizing a possibly non-square $k \times l$ multiplier from $b \times c$ component multipliers or multiply-add modules, demonstrating that in many cases, the use of rectangular components leads to speed and cost benefits.*

KEYWORDS: Architecture optimization; Aspect ratio; Digital computer arithmetic; Divide and conquer; Fast multiplication; Modular design; Multiplier; Multiply-add module; Squarer.

I. INTRODUCTION

Multiplication is the most important arithmetic operation after addition/subtraction. Given that multipliers tend to be slower and more complex than adders/subtractors, in many applications the performance and implementation cost are dictated to a large degree by the design used for multipliers. It is thus not surprising that from the early days of digital computers, proposals for fast multipliers and schemes for designing such circuits in a cost-effective manner [14], [3]. The attraction of logarithmic arithmetic [12] is also primarily due to its support for fast multiplication (as well as division, squaring, and square-rooting).

The fastest multipliers tend to be expensive in terms of chip area (in custom designs) or logic and interconnect usage (in FPGA-based designs). Thus, alternate designs are continually sought that either provide a reasonable speed at lower cost or else provide high throughput for applications that involve a large number of independent multiplications. Serial-parallel and array multipliers are examples of such designs that provide cost-effectiveness and high throughput, respectively [11].

Recursive multiplication often does not lead to the fastest possible circuit, but it provides some advantages that make it a useful addition to the repertoire of a digital system designer. In this scheme, small multipliers are used as building blocks for synthesizing a multiplier of desired size. One immediately obvious application in hardware is when wider multipliers are needed compared with those that are built-in on modern FPGAs [15]. Another example of utility is when several parallel subword multiplications should be performed using the same basic hardware [5]. A software-implemented example is multiplying extremely wide numbers of the kind encountered in cryptography.

The conceptually simplest recursive scheme for building a $k \times l$ multiplier is when k and l are powers of 2 and, at each recursion step, we divide the operands into halves. The recursion can conceptually continue down to the bit level, but we often stop when a sufficiently small operand width has been reached, with efficient available hardware designs or the possibility of using lookup tables. Even though some published examples of the use of small multipliers in synthesizing recursive multipliers do use non-square modules (e.g., 4×8), a general discussion of square versus rectangular component multipliers is lacking in the literature.

The rest of this paper is organized as follows. The basic ideas of recursive multiplication and its hardware implementation with smaller multipliers or additive multiply modules appear in Section II. A discussion of hardware designs with square modules sets the stage for our coverage of non-square or rectangular modules in Section III. Several examples that probe the corner cases of our general method and its associated analysis are presented in dot notation. Section IV starts with a recap of the potential hardware, energy, and latency reductions offered by rectangular component multipliers and ends with our conclusions and directions for further research.

II. Recursive Multiplication

A. Basic Theory

Recursive multiplication uses a divide-and-conquer strategy. We consider only unsigned multiplication in the following discussion. To multiply a k-bit number by an l-bit number using $b \times c$ component multipliers, we proceed as follows. Assume for simplicity that $k = gb$ and $l = hc$, that is, k and l are divisible by b and c, respectively. The process begins with performing gh multiplications on b-bit and c-bit chunks of the two operands, producing gh partial products of width $b + c$. The rest of the process is a multi-operand addition problem to combine the gh partial products into a single final product of width $k + l$. Because the gh partial products do not span the entire $(k + l)$-bit width of the final product, the height of the partial-products matrix is often significantly less than gh.

B. Multi-Operand Addition

Once the partial products have been obtained from component multipliers, they must be added together to form the final product. Let's focus on square multipliers first. As shown in Fig. 2, doubling the multiplier width, from $b \times b$ to $2b \times 2b$, entails a 3-operand addition, which can be performed by one level of carry-save addition, followed by a normal carry-propagate addition. Tripling the width requires 5-operand addition, composed of a 3-level CSA and a CPA. Quadrupling of the width requires a 4-level CSA and a CPA. In general, CSA tree depth increases as the logarithm of the height $2g - 1$ of the partial-products matrix.

Alternatively, specially designed $(n; 2)$-compressors can be used that convert a column of bits of height n to 2 bits, one in the same column and one in the next higher column (worth twice as much), along with a set of transfer bits into one or more higher columns. Various carry configurations and multiplicities are feasible [11].

With non-square final or component multipliers, the height of the partial-products matrix will change, as we will see in Section III, leading to simpler and faster accumulation in many cases. These circuit and time savings constitute the main points of this paper.

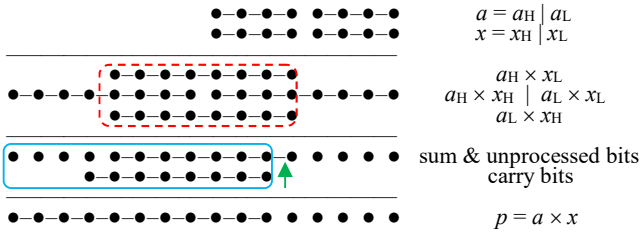

Fig. 1. An 8 × 8 multiplier built from 4 × 4 component multipliers.

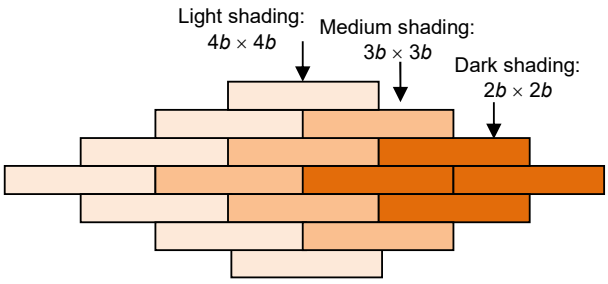

Fig. 2. Alignment of partial products in a $gb \times gb$ multiplier built of $b \times b$ component multipliers.

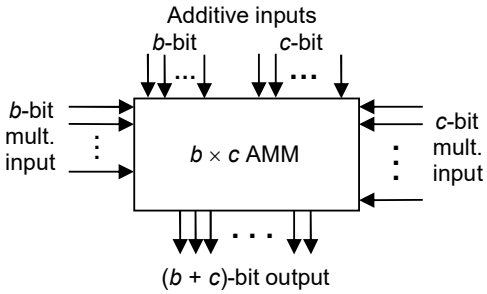

Fig. 3. Block diagram of a $b \times c$ additive multiply module.

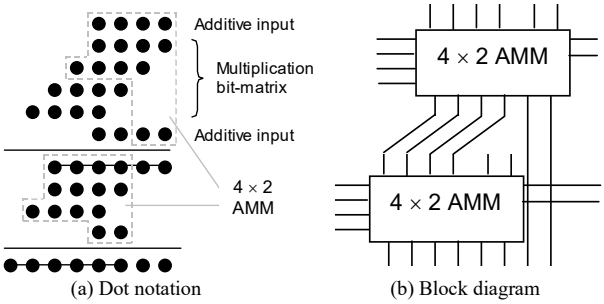

Fig. 4. A 4 × 4 multiplier built of 4 × 2 AMMs.

C. Additive Multipliers

An additive multiply module (AMM) combines multiplication and addition by computing the $(b + c)$-bit result, where a and y (x and z) are b-bit (c-bit) numbers.

$$p = ax + y + z$$

Except at the final circuit level, where the $(b + c)$-bit output of the block forms part of the final product p, the AMM output bits are divided into b-bit and c-bit parts, with each part connected to an additive input of an AMM block at the next level. An example of a 4 × 4 multiplier built of 4 × 2 AMMs is depicted in Fig. 4, where the dot notation in part (a) justifies the circuit diagram in part (b). The circuit in Fig. 4(b) can be converted to a 4 × 4 AMM if we connect 4-bit additive inputs to the 4-bit input at the top and to the two 2-bit inputs at the top right of the two blocks.

D. The Special Case of Squaring

Squarers tend to be simpler and faster than multipliers of the same width. Thus, in applications calling for squaring operations, it may be more cost-effective to use specialized squaring circuits instead of using multipliers with both inputs tied to the same value.

Considering the example in Fig. 1 as a squaring, rather than multiplication, operation, we note that setting $a = x$ will make $a_H \times x_L$ and $a_L \times x_H$ equal, allowing us to replace two rows of dots in the middle section of Fig. 1 with a single row that is the 1-bit left-shifted version of one of the two identical lines. This modification will eliminate the need for the CSA, with the summation performed by just an 11-bit CPA, resulting in a faster and simpler circuit. Also, we need just 3 components, instead of 4, with two of them being component squarers and one being a component multiplier.

III. Designs with Rectangular Modules

Analysis of multipliers built of rectangular modules entails complicated combinatorial analyses to determine the height of the partial-products matrix, as we will see later. It is thus prudent to begin with a couple of examples for getting a feel for the issues involved.

Consider an example, depicted in Fig. 5, having $b = 3$, $c = 4$, $g = 4$, and $h = 3$. In building the 12×12 multiplier, we form 12 partial products, which must be added. These partial products can be rearranged to form only 6 lines, as shown near the bottom of Fig. 5. The same 12×12 multiplier can be built from 2×6 components, also requiring 12 modules and the addition of 12 partial products, rearrangeable into 7 lines (Fig. 6). A natural question is which of the components, 4×4 or 3×3, leads to a better 12×12 multiplier in terms of speed, chip area, and other relevant performance metrics.

Note that the shape of the partial-products matrix and the total number of rows after row-merging is a function of g, h and the ratio b/c and not the particular values of b and c. The same basic shape as Fig. 5 will be observed for a 24×24 multiplier built of 6×8 modules, or, more generally, for a $12j \times 12j$ multiplier built of $3j \times 4j$ modules. Similarly, the partial-products matrix and final number of rows for a $12j \times 12j$ multiplier built of $2j \times 6j$ modules will look like the one in Fig. 6.

Here is a general analysis for a non-square $gb \times hb$ multiplier built from non-square $b \times c$ component multipliers. Without loss of generality, we assume $g \geq h$. Let bit positions of a be indexed from 0 to $gb - 1$, and divide it into b-bit segments beginning at bit positions ib $(0 \leq i \leq g - 1)$ and ending at $(i + 1)b - 1$. Similarly, bits of the multiplier x are indexed from 0 to $hc - 1$, with its c-bit segments beginning at positions jc $(0 \leq j \leq h - 1)$ and ending at $(j + 1)c - 1$. All partial products are $b + c$ bits wide and we can divide them into two groups of h rows and $g - 1$ rows.

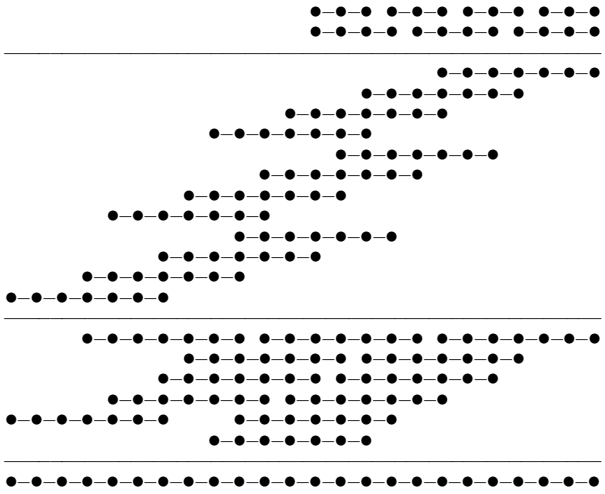

Fig. 5. Recursive 12×12 multiplier built of 3×4 modules.

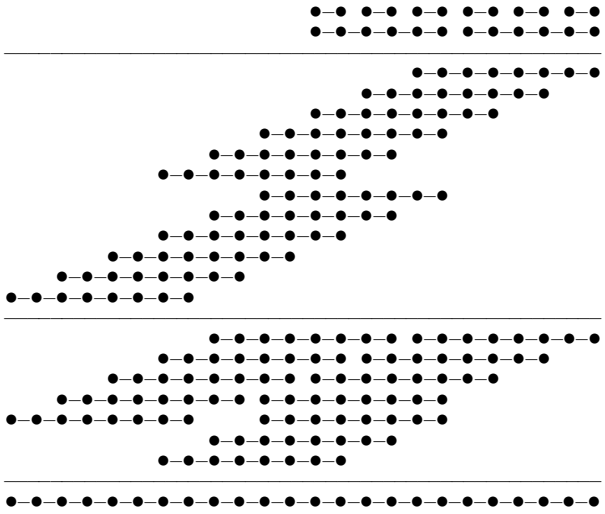

Fig. 6. Recursive 12×12 multiplier built of 2×6 modules.

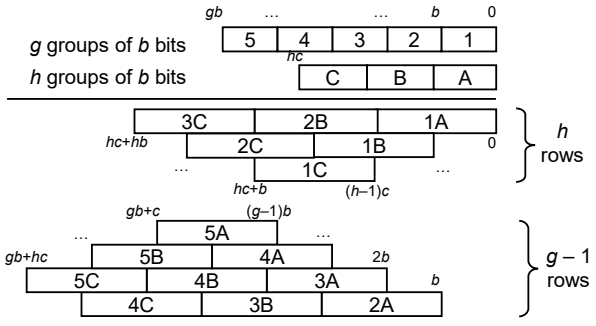

Fig. 7. Recursive $gb \times hc$ multiplier built of $b \times c$ modules.

The arrangement of the gh partial products in Fig. 7 establishes a height upper bound of $g + h - 1$. The remaining problem is whether the height can be reduced further. It is easy to see from the diagram that the top h rows cannot be compressed within themselves. The bottom $g - 1$ rows may be compressible, but we will return to this kind of compression shortly.

A clear possibility for height reduction is the matching of the staircaselike left boundary of the top h rows and that at the right side of the bottom $g - 1$ rows. For example, if the staircase edge at $(g - 1)b$ of the lower staircase matches the edge $hc + b$ of the upper one, one row can be taken out by shifting up the lower rows to align the boxes 5A and 1C in Fig. 7. More generally, if the staircase edge at $(g - 1)b$ of the lower staircase matches the edge $hc + bx$ of the upper one, we get the height $g + h - 1 - x$. The fact that x cannot exceed h, along with the inequality $(g - 1)b \geq xb + hc$, leads to the reduction $x^{\max} = \max[0, \min(h, g - 1 - \lceil hc/b \rceil)]$.

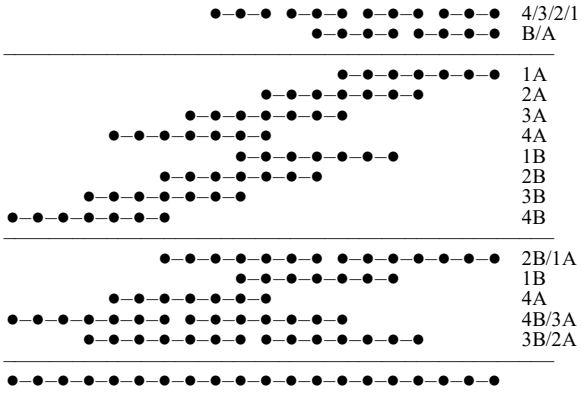

Fig. 8. Recursive 12 × 8 multiplier built of 3 × 4 modules.

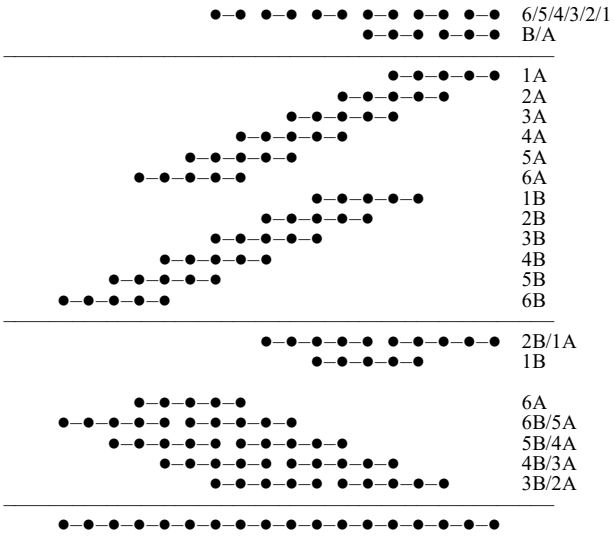

Fig. 9. Recursive 12 × 6 multiplier built of 2 × 3 modules.

Let us explore the preceding bound by means of a simple example (Fig. 8). Here, we have $b = 3$, $c = 4$, $g = 4$, $h = 2$, leading to $x \leq \max[0, \min(2, 4 - 1 - \lceil 2\times4/3 \rceil)] = 0$. The overlap of 2 bits among all 5 rows near the bottom of Fig. 8 confirms our analysis that no compression is possible. Consider now another example with $b = 2$, $c = 3$, $g = 6$, $h = 2$. In this case, as confirmed by Fig. 9, we have $x \leq \max[0, \min(2, 6 - 1 - \lceil 2\times3/2 \rceil)] = 2$.

Before returning to the topic of possible additional compression solely within the bottom $g - 1$ rows, let us further verify our results by considering some special cases. For square multipliers built of square components ($b = c$, $g = h$), we get $x^{\max} = 0$ and the height $2g - 1$ of the partial-products matrix is irreducible, as expected. For non-square multipliers synthesized from square components ($b = c$, $g > h$), $x^{\max} = \min(h, g - 1 - h)$, leading to the compressed height of $g - 1$ if $g \leq 2h$, or $2h$ if $g > 2h$. The latter outcome for the case $g > 2h$ needs to be amended, as we will see shortly. Now, consider the bottom $g - 1$ rows in Fig. 7. It is conceivable that for g sufficiently larger than h, the bottommost rows will have no overlap with the topmost rows, leading to further compression within these rows. For this to happen, the leftmost edge of the lowermost row, located at index $(h + 1)b + hc$ should be at or to the right of the edge $(g - y)b$ of the staircase to the right, in order to get y rows of reduction. Thus, $y^{\max} = \max[0, \min(h, g - h - 1 - \lceil hc/b \rceil)]$.

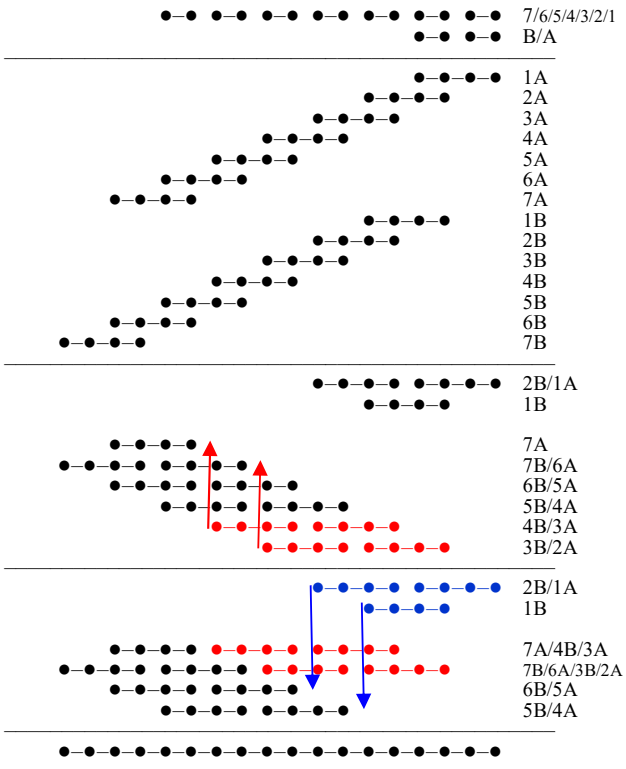

Fig. 10. Recursive 14 × 4 multiplier built of 2 × 2 modules.

We use an example to get a feel for the nature of this additional row compression (Fig. 10). Here, we have $b = 2$, $c = 2$, $g = 7$, $h = 2$, leading to $y \leq \max[0, \min(2, 7 - 2 - 1 - \lceil 2\times 2/2 \rceil)] = 2$. The compressed version, near the bottom of Fig. 10, shows a reduction of 2 rows within the lower set of rows. It turns out that an additional 2 rows can be removed by merging the upper and lower sets of rows, and this reduction does not interfere with the other one. We do not have a precise formulation for the conditions under which the two kinds of reduction interfere with each other, so for now, one has to determine the amount of each reduction separately, using our formulas, and then check to find the total reduction $z^{\max} \leq x^{\max} + y^{\max}$.

Given that the compression within the lower set of rows occurs only when $g > \lceil hc/b \rceil + h + 1$, the resulting uncertainty in the amount of reduction can be avoided by ensuring that g isn't much greater than h. This can be done for example, by switching the values of b and c, given that $b \times c$ and $c \times b$ multipliers have the same cost.

IV. Conclusion

We have studied the implications of using non-square or rectangular component multipliers for building both square and rectangular multipliers recursively. The key to simplifications reviewed here and the attendant area, power, and latency reductions is the skewed partial-products dot matrix, as exemplified by Fig. 11, which hints at the possibility of reducing the matrix height during the multi-operand addition phase.

We derived closed-form formulas for the total number of partial-products rows may must be compressed as part of the multiplication process, along with conditions under which the formulas are valid. We observed that the special cases where the formulas fail and additional computational steps are needed to deduce the height of the partial-products matrix are of limited practical significance. Additional work is required to see if formulas that work in all cases can be derived.

Actual circuit realizations or detailed circuit modeling is needed to derive area, power, and latency reductions resulting from rectangular component multipliers.

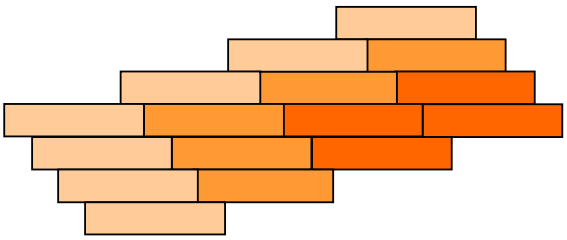

Fig. 11. The counterpart of Fig. 2 when the component multipliers are rectangular, rather than square.

One key conclusion is that the choice of aspect ratio for component multipliers will impact the overall design's regularity and complexity, as well as its latency. Our results also have a bearing on the tradeoffs involved in choosing components to optimize a design for specific implementation technology, such as LUT-based FPGAs. In the latter context, 2×4 or 3×3 component multipliers may be preferable to other sizes, if 6-input LUTs are available. Such specific options for realization constitute fruitful areas for further investigation.

References

[1] K. Biswas, P. Mokrain, H. Wu, and M. Ahmadi, "Truncation Schemes for Recursive Multipliers," *Proc. 39th Asilomar Conf. Signals, Systems, and Computers,* 2005, pp. 1177-1180.

[2] K. Biswas, H. Wu, and M. Ahmadi, "Fixed-Width Multi-Level Recursive Multipliers," *Proc. 40th Asilomar Conf. Signals, Systems, and Computers*, 2006, pp. 935-938.

[3] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, Vol. 34, pp. 346-356, May 1965.

[4] A. N. Danysh and E. E. Swartzlander, "A Recursive Fast Multiplier," *Proc. 32nd Asilomar Conf. Signals, Systems, and Computers*, 1998, Vol. 1, pp. 197-201.

[5] J. Fridman, "Sub-word Parallelism in Digital Signal Processing," *IEEE Signal Processing Magazine*, Vol. 17, No. 2, pp. 27-35, March 2000.

[6] C. Ghest, "Multiplying Made Easy for Digital Assemblies," *Electronics*, Vol. 44, pp. 55-61, 22 November 1971.

[7] G. J. Hekstra and R. Nouta, "A Fast Parallel Multiplier Architecture," *Proc. IEEE Int'l Symp. Circuits and Systems*, 1992, Vol. 5, pp. 2128-2131.

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

[9] J. Kim and E. E. Swartzlander, "Improving the Recursive Multiplier," *Proc. 34th Asilomar Conf. Signals, Systems, and Computers*, 2000, Vol. 2, pp. 1320-1324.

[10] N. Nedjah and L. de Mercedo Mourelle, "Fast Less Recursive Hardware for Large Number Multiplication Using Karatsuba-Ofman's Algorithm," *Computer and Information Sciences*, LNCS Vol. 2869, Springer, 2003, pp. 43-50.

[11] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2nd ed., 2010.

[12] B. Parhami, "A Review of Logarithmic Arithmetic," in preparation.

[13] C. C. Stearns and P. H. Ang, "Yet Another Multiplier Architecture," *Proc. IEEE Custom Integrated Circuit Conf.*, 1990, pp. 24.6.1-24.6.4.

[14] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electronic Computers*, Vol. 13, pp. 14-17, February 1964.

[15] Xilinx, "Using Embedded Multipliers in Spartan-3 FPGAs," Application Note, May 13, 2003.