

Exploring realizations of large integer multipliers using embedded blocks in modern FPGAs.

Shreesh Srinath

Abstract

An efficient design methodology and a systematic approach for the implementation of multiplication for unsigned large integers, using small-size asymmetric embedded multipliers is presented. Three different approaches are explored for the design. A general architecture of the multiplier is proposed and a set of equations is derived to aid in the realization. The inputs of the multiplier are split into several segments leading to an efficient utilization of the small-size embedded multipliers and a reduced number of required addition operations. The equations target modern FPGA platforms such as Xilinx Virtex-5 which provide 25 x 18 sized multipliers. Finally the implementation of the three designs are carried out and the results show that the proposed method is best in area and timing.

1.0 Introduction

Multiplication functions constitute the kernel of many real-life applications. They are used extensively in applications such as digital signal processing, image processing, cryptography and multimedia [1,2,3]. Recent computing oriented FPGAs feature embedded DSP blocks including small embedded multipliers. Achieving efficient realization of multiplication may have significant impact on the specific application in terms of speed, power dissipation and area.

FPGA vendors are now offering hardwired multipliers as one of the resources available to designers. Examples could be that of Xilinx Spartan-3 Family which includes 104 on-chip 18x18 multipliers and Xilinx Virtex-5 & 6 Family which include 25x18 multipliers. Optimized realizations of large multipliers of large integer multipliers using such blocks are studied in [4,5,6]. A paper-and-pencil analysis of FPGA peak floating-point performance [9] clearly shows that DSP blocks are a relatively scarce resource when one wants to use them for accelerating multiplications. This project aims to study different approaches to implement large integer multipliers on asymmetric DSP blocks in an efficient manner in terms of both timing and area.

2.0 Related Work

In [4,5], the authors present an efficient design methodology and systematic approach for implementation of multiplication and squaring functions. They propose a general architecture and a set of equations are derived to aid in realization. The method used is that of the “Divide and Conquer Algorithm” [7] with efficient organization of partial products. The symmetric embedded block considered is of size “ $n \times n$ ” and operands of size “ k ” such that $(n \times (m-1)) < k \leq (n \times m)$. The Table 1, below gives the sizes, in bits, of the partial products in the multiplication expression of two operands “ X and Y ”.

Table 1: Sizes, in bits, of the partial products in expression (5)

Partial products	Number of bits
$X_{m-1} \times Y_{m-1}$	$2 \times (k - (m - 1) \times n)$
$X_i \times Y_i$, where $i = m - 2, m - 3, \dots, 1, 0$	$2 \times n$
$X_{m-1} \times Y_i$ or $Y_{m-1} \times X_i$, where $i = m - 2, m - 3, \dots, 1, 0$	$(k - (m - 1) \times n) + n$
$X_i \times Y_j$ or $Y_i \times X_j$, where $i \neq j$ and $i, j = m - 2, m - 3, \dots, 1, 0$	$2 \times n$

The authors then look at timing and area efficient organization of the additions of partial products including the method of deferred parallel carry addition of partial products in which the set of carry bits generated from various levels of partial product additions are combined and processed later.

In [6], the authors note the use of *Karatsuba-Ofman* algorithm [8], and present detailed methodology of splitting the operations by use of the algorithms has been proposed to implement large multiplication using less number of DSP blocks which again target the “ $n \times n$ ” basic embedded multipliers. The authors mention that no reference to Karatsuba-Ofman algorithm for integer multiplication in was found in FPGA literature.

3.0 Asymmetric embedded blocks

The prior studies deal with the implementation of large multipliers using symmetric “ $n \times n$ ” embedded blocks. The goal of this project is to study the previous methods of [4,5,6] and develop a set of equations to assist the design and implementation of a large multiplier using asymmetric “ $m \times n$ ” embedded multipliers which are now available in the Xilinx Virtex-5 and 6 families. The DSP48E block in the Xilinx Virtex-5 & 6 provides 25 x 18 sized hardwired multipliers. The approach is different to prior work and is novel as it deals with the asymmetric hardwired multipliers. The following section provides an example of the approach and discusses the alternatives to the proposed approach. Sections 4.x provide the design equations which implement the proposed approach. [More on the other sections description]

3.1 Example

Consider the multiplication of two numbers **X** and **Y** both of size **k** bits, which are split into the following chunks of **m** and **n** (**m < n**) as shown below.

$$\mathbf{X} = [x_2 \ x_1 \ x_0] \text{ and } \mathbf{Y} = [y_3 \ y_2 \ y_1 \ y_0];$$

$$\begin{aligned} \mathbf{X} &= (2^{2n}.x_2 + 2^n.x_1 + x_0); \\ \mathbf{Y} &= (2^{3m}.y_3 + 2^{2m}.y_2 + 2^m.y_1 + y_0); \end{aligned}$$

Consider the Multiplication **Z = X*Y**. The computation of **Z** is implemented using embedded multipliers of size **m x n**. Substituting X and Y we get,

$$\mathbf{Z} = (2^{2n}.x_2 + 2^n.x_1 + x_0) * (2^{3m}.y_3 + 2^{2m}.y_2 + 2^m.y_1 + y_0).$$

$$\begin{aligned} \mathbf{Z} &= 2^{2n}.(x_2.y_0) + 2^{2n}.(x_1.y_0) + (x_0.y_0) \\ &+ 2^{(2n+m)}.(x_2.y_1) + 2^{(n+m)}.(x_1.y_1) + 2^{(m)}.(x_0.y_0) \\ &+ 2^{(2n+2m)}.(x_2.y_2) + 2^{(n+2m)}.(x_1.y_2) + 2^{(2m)}.(x_0.y_2) \\ &+ 2^{(2n+3m)}.(x_2.y_3) + 2^{(n+3m)}.(x_1.y_3) + 2^{(3m)}.(x_0.y_3) \end{aligned}$$

Replacing the terms with a(1,2,3), b(1,2,3), c(1,2,3) & d(1,2,3) as below:

$$\begin{aligned} \mathbf{Z} &= a_1 + a_2 + a_3 \\ &+ b_1 + b_2 + b_3 \\ &+ c_1 + c_2 + c_3 \\ &+ d_1 + d_2 + d_3. \end{aligned}$$

Each product term $a(1,2,3)$, $b(1,2,3)$, $c(1,2,3)$ & $d(1,2,3)$ is evaluated on a $m \times n$ multiplier and the result is of length $m+n$ bits. The products then obtained are added by grouping the terms and the final result of the multiplication is obtained. The product terms can be grouped to generate partial products which then are added to provide the value of the result. Three choices are available for the grouping of the terms : 1.) Horizontal 2.) Vertical and 3.) Diagonal. The following sections briefly describes these options in brief and an implementation cost is evaluated for each option to determine the best choice.

In each of the implementations we employ the scheme of deferred carry addition described in [5] with the caveat that, we allow for overlap of the carry bit from any stage upto two times as the hardware required to handle these cases is simple and any number above two we drop the optimization. The case where the overlaps occur can be predetermined by solving a set of equalities in which we equate the positions of each carry bits and check for no more than two solutions. This is explained when general equations are provided for the proposed approach in section 4.0.

3.1.1 Grouping Horizontally

The terms when grouped horizontally we obtain four partial-products as shown below:

$$\begin{aligned} \text{PP1: } & (a_1 + a_2 + a_3) \\ \text{PP2: } & (b_1 + b_2 + b_3) \\ \text{PP3: } & (c_1 + c_2 + c_3) \\ \text{PP4: } & (d_1 + d_2 + d_3) \end{aligned}$$

The partial products PP(1...4) are then summed to get the result Z . The equations above are converted to constraint graphs with the organization for best timing and the resource requirements for each stage is then calculated to obtain the cost of the implementation. To estimate the cost of the implementation of the adders and the multipliers of each stage is assumed to be 1 unit. This assumption is true for initial evaluations and can be further improved with larger adders as the bit size of X and Y increase.

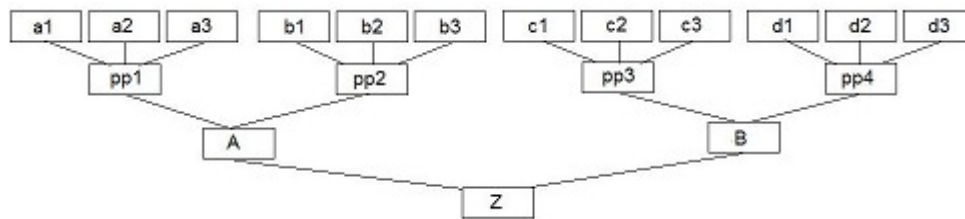


Figure 1

The cost of the implementation above in terms of additions and multiplications are given as below:

Multiplications	12
Additions	12
Total cost (additions + multiplications)	24

Table 2

The result is obtained after 7 clock cycles with the above organization.

The resource requirements are shown below:

Number of Adders	Size in bits
8	m
2	2n
1	3n
1	3m+2n

Table 3

The number of multipliers shown below are assuming a multistage design with no pipelining.

Number of Multipliers	Size in bits
12	m x n

Table 4

3.1.2 Grouping Vertically

The terms when grouped vertically we obtain three partial-products as shown below:

$$PP1: (a1 + b1 + c1 + d1)$$

$$PP2: (a2 + b2 + c2 + d2)$$

$$PP3: (a3 + b3 + c3 + d3)$$

The partial products PP(1...3) are then summed to get the result **Z**. The equations above are converted to constraint graphs with the organization for best timing and the resource requirements for each stage is then calculated to obtain the cost of the implementation as shown below.

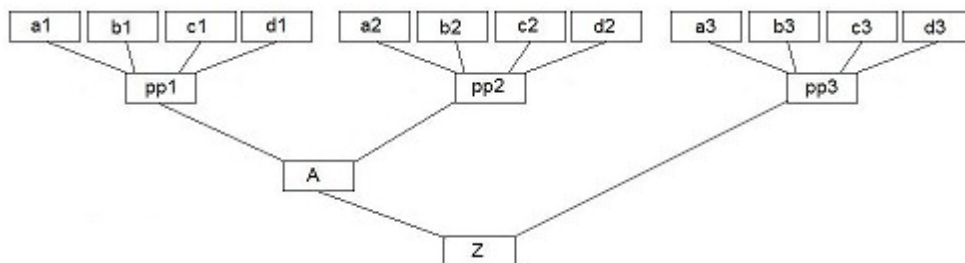


Figure 2

The cost of the implementation above in terms of additions and multiplications are given as below:

Multiplications	12
Additions	12
Total cost (additions + multiplications)	24

Table 5

The resource requirements are shown below:

Number of Adders	Size in bits
9	n
2	4m
1	3m+2n

Table 6

The number of multipliers shown below are assuming a multistage design with no pipelining.

Number of Multipliers	Size in bits
12	m x n

Table 7

The result is obtained after 7 cycles as the previous case. From the initial evaluations of the two design choices we can know that the resource requirements of the vertical case is much larger than that compared to the horizontal case. Hence, translating the equations to constraint graphs and tabulating the resource requirements prove to be useful to assist the designer to eliminate choices and narrow the design space exploration.

3.1.2 Grouping Diagonally

The terms when grouped diagonally we obtain six partial-products as shown below:

- PP1: (a1)
- PP2: (b2 + a1)
- PP3: (c1 + b2 + a3)
- PP4: (d1 + c2 + b3)
- PP5: (d2 + c3)
- PP6: (d1)

The partial products PP(1...6) are then summed to get the result **Z**. The equations above are converted to constraint graphs with the organization for best timing and the resource requirements for each stage is then calculated to obtain the cost of the implementation as shown below.

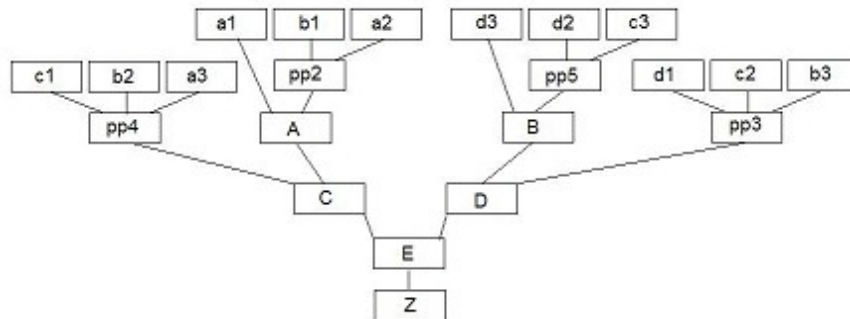


Figure 3

The cost of the implementation above in terms of additions and multiplications are given as below:

Multiplications	12
Additions	5
Total cost (additions + multiplications)	17

Table 8

The resource requirements are shown below:

Number of Adders	Size in bits
2	$m+n$
2	$2(m+n)$
1	$3m$

Table 9

The number of multipliers shown below are assuming a multistage design with no pipelining.

Number of Multipliers	Size in bits
12	$m \times n$

Table 10

The result is obtained after 5 cycles. The number of addition operations drastically reduce as we go to the grouping of diagonally. This is because although it appears that addition operations are required to generate $PP(1\dots6)$ the addition operations are saved as the individual terms of each partial-product $PP(1\dots6)$ reduces to a mere concatenation. Each term constituting a partial-product is of length $m+n$ bits and is shifted such that there exists no overlap between the terms. This is illustrated as below considering the third partial product, $PP3$. The partial product $PP3 = x_2y_2 + x_1y_1 + x_0y_0$. The arrangement (concatenation) of each term is as shown below.

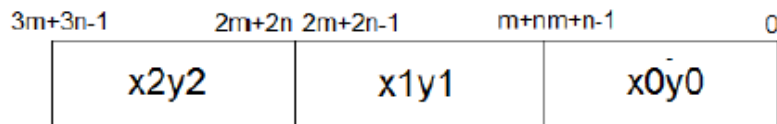


Figure 4

The other interesting property of this grouping is that the number of stages of additions increase logarithmic and the size of the adder required at each stage as well increases logarithmically with stage q requiring a largest adder of size $q(m+n)$ bits and upto a maximum number of 2. This grouping is hence chosen for large scalable design and it provides for the best timing and saves on the area as the size of the computation increases. Hence, the grouping of the terms diagonally and the organization of the terms is explored and design equations are provided for the general case in the following section.

4.0 Design approach for the general case

For the general case of the multiplier design. The first step is to determine the number of m -sized and n -sized chunks. The equations below help in determining these numbers.

Eq 1 : Number of n -sized chunks = $\text{ceiling}(k/n) = p_1$

Eq 2 : Number of m -sized chunks = $\text{ceiling}(k/m) = p_2$

The last chunks of the numbers X (split into n -sized chunks) and Y (split into m -sized chunks) each contain $(n \cdot p1 - k)$ and $(m \cdot p2 - k)$ number of zeros. Hence using the above two equations X and Y can be written as below.

$$X = [X_{p1-1} \ X_{p1-2} \ \dots \ X_0]_{2^n}$$

$$Y = [Y_{p2-1} \ Y_{p2-2} \ \dots \ Y_0]_{2^m}$$

$$X = 2^{(n \cdot (p1-1))} X_{p1-1} + 2^{(n \cdot (p1-2))} X_{p1-2} + \dots + X_0$$

$$Y = 2^{(m \cdot (p2-1))} Y_{p2-1} + 2^{(m \cdot (p2-2))} Y_{p2-2} + \dots + Y_0$$

Hence, we now evaluate $Z = X \cdot Y$ substituting the values above as below:

$$Z = ([X_{p1-1} \ X_{p1-2} \ \dots \ X_0]_{2^n}) * ([Y_{p2-1} \ Y_{p2-2} \ \dots \ Y_0]_{2^m})$$

$$Z = (2^{(n \cdot (p1-1))} X_{p1-1} + 2^{(n \cdot (p1-2))} X_{p1-2} + \dots + X_0) * (2^{(m \cdot (p2-1))} Y_{p2-1} + 2^{(m \cdot (p2-2))} Y_{p2-2} + \dots + Y_0)$$

We now consider the aligning of the partial products and the addition stages in order to reduce the combinational logic delay and area. According to the expanded X and Y as shown above each product term constituting Z is as shown below.

$$PP_{ij} = X_{p1-i} * Y_{p2-j} = (2^{(n \cdot (p1-i) + (m \cdot (p2-j)))}) X_{p1-i} * Y_{p2-j}$$

where $(0,0) \leq i,j \leq (p1-1, p2-1)$ and the length of each partial result is $m+n$ bits.

The number of the partial products obtained is given by the equation below:

$$\text{Eq 3 : } (p2 + p1 - 1)$$

Each term is computed diagonally and enumerated as below (with the bounds of the terms) :

- 1.) $X_{p1-1} * Y_0$ $\{n \cdot (p1) + m, n \cdot (p1-1)\}$
- 2.) $X_{p1-1} * Y_1, X_{p1-2} * Y_0$ $\{n \cdot (p1) + 2m, n \cdot (p1-2)\}$
- 3.) $X_{p1-1} * Y_2, X_{p1-2} * Y_1, X_{p1-3} * Y_0$ $\{n \cdot (p1) + 3m, n \cdot (p1-3)\}$
- ⋮
- ⋮
- ⋮
- $p1+p2-3$.) $X_2 * Y_{p2-1}, X_1 * Y_{p2-2}, X_0 * Y_{p2-3}$ $\{m \cdot (p2) + 3n, m \cdot (p2-3)\}$
- $p1+p2-2$.) $X_1 * Y_{p2-1}, X_0 * Y_{p2-2}$ $\{m \cdot (p2) + 2n, m \cdot (p2-2)\}$
- $p1+p2-1$.) $X_0 * Y_{p2-1}$ $\{m \cdot (p2) + n, m \cdot (p2-1)\}$

The number of adding stages is obtained as **ceiling(log₂ (p₂ + p₁ - 1))**. To optimize the additions of the partial products two optimizations targets are considered : timing-oriented and area-saving oriented. Using smaller adders would help both the performance and area considerations. The addition of the partial products operations are illustrated as below.

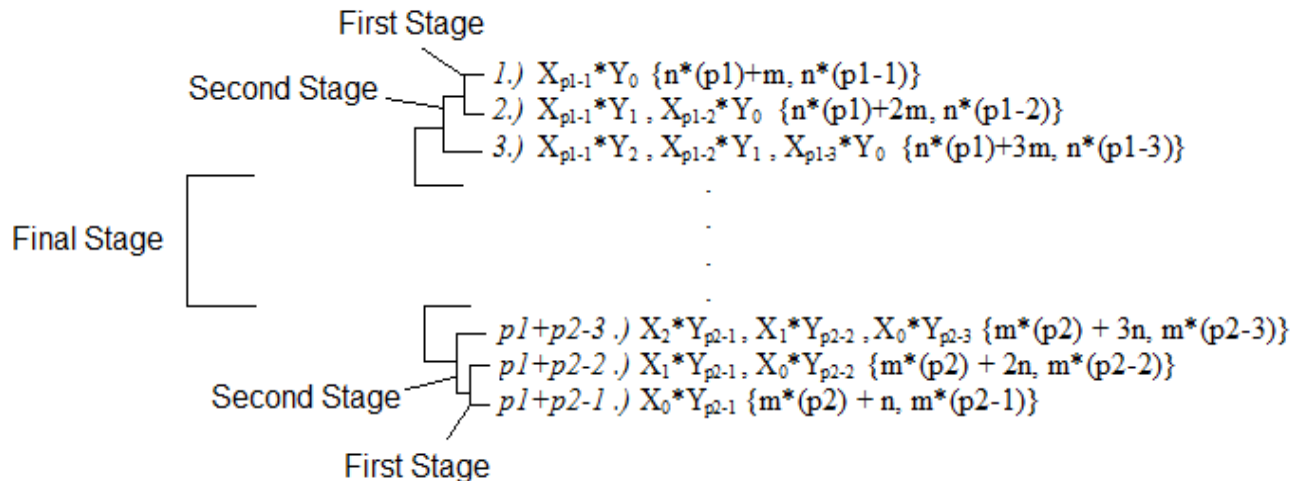


Figure 5

Two cases arise depending on the number of partial products i.e. **(p₂ + p₁ - 1)** the number being even or odd. In case of even number of partial products the addition stages are simple and compressed two at a time as shown above till the final stage i.e. **Q = ceiling(log₂ (p₂ + p₁ - 1))**. The last stage **Q** would a simple reduction of two partial sum results from the **Q-1 th** stage. In case the number of products are odd. The last but one stage **Q-1 th** would be a reduction of a partial product and an intermediate result of the previous stages and the last stage **Q** would be a compression of this result with another partial product of the **Q-1 th** (from bottom half) to obtain the final result. The size of the adder (explained below) of the **Q-1 th** and the **Q th** stage when the total number of partial products is odd would be the same.

The interesting property of the above addition stages is the size of the adder in each addition stage. The size required for the reduction at each level happens to increase logarithmically with the increase in the number of the levels. This property would favor the scalability of the adder stages to a larger numbers as compared to the other design alternatives. The adder required at each stage is as shown below and a maximum number at each stage is utmost two.

- Stage 1: Largest adder – m+n
- Stage 2: Largest adder – 2(m+n)
- Stage 3: Largest adder – 3(m+n)
-
-
-
-
-
- Stage Q: Largest adder – Q(m+n)

Figure 6

The final stage of addition is the addition of the partial result of the addition stage and that of the deferred carry of each stage. The position of the deferred carry of each level of addition can be enumerated as below:

Addition Stage	Carry positions (from upper half)	Carry positions (from bottom half)
1	$n*(p1)+m$	$m*(p2) + n$
2	$n*(p1)+2m$	$m*(p2) + 2n$
3	$n*(p1)+3m$	$m*(p2) + 3n$
.	.	.
.	.	.
.	.	.
.	.	.
Q	$n*(p1)+Qm$	$m*(p2) + Qn$

Table 11

As explained in section 3.0 the bit position of the carries are equated and the system of equations are checked such that no more than two solutions exist and if so the deferred parallel carry addition optimization can be applied else the optimization is dropped for the stages which overlap till the overlap drops back to two stages.

5.0 Implementation examples on Xilinx Virtex-5 platform

Using the aid of the design equations from section 4.0 and the discussion of section 3.0 three designs were implemented on the Xilinx Virtex-5 FPGA platform. The three multipliers considered are :

- 1.) Baseline Horizontal Multiplier with no deferred carry additions : This is chosen as the baseline and is the design which is the basic and normal implementation of a multiplier.
- 2.) Horizontal Grouping with deferred carry additions : This multiplier design uses the above organization with deferred carry additions optimization.
- 3.) Diagonal Grouping with deferred carry additions: The multiplier design considered here is based on the equations of section 4.0.

All the three multiplier designs considered are multistage multipliers based on the description of the examples given in the previous sections. The multiplier designs take advantage of the built-in modules available on the Xilinx Virtex-5 XC5VLX110-2FF676 FPGA using the Xilinx ISE 10.1.03 tool suite, but should also be applicable to other high-performance FPGA platforms. This particular device includes 17,28 slices (each slice has four 6-input lookup tables (6-LUTs) and four flip-flops), 256 18-Kb Block RAMs, and 64 high-performance DSP48E blocks that support multiplication, multiply-accumulate, three-input addition, and other functions. The DSP48E blocks of the device provide the asymmetric building blocks capable providing 24 x 17 sized multiplication. The results of the implementation are as tabulated below.

Design	Number of Slice Registers	Number of LUTs	Number of DSP48Es	Frequency (Mhz)
Baseline	498	736	12	147.22
Horizontal	506	601	12	239.2
Diagonal	517	551	12	254.05

Table 12

The multiplier designs are capable of handling 52-68 bits of multiplications. The implementation results confirm the design choices favoring the diagonal grouping. The design can be further pipelined to yield faster a multiplier.

6.0 Conclusions

The focus of this paper is to provide designers with an optimized solution to realize large integer multipliers using small-size optimized asymmetric “ $m \times n$ ” sized embedded multipliers. The approach presented here is based on the divide-and-conquer strategy with a set of efficient schemes for realizing the required additions. Three different groupings of the partial products are analyzed for implications of area and timing. The proposed addition schemes for diagonal groupings provides the best area-efficient and timing-oriented realization of the multiplier. Transforming the given design equations to constraint-graph is a useful tool to analyze the resource requirements and hence decide the schedule for best area or best timing as per the design constraints. The multiplier designs can further be improvised for pipelining. The design equations aid the designer and the transformation to constraint-graph could be useful to explore the design space.

References

- [1] Walters, E. III, Arnold, M.G., and Schulte, M.J.: “Using truncated multipliers in DCT and IDCT hardware accelerators”. Proc. SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XIII, San Diego, California, August 2003, pp. 573–584.
- [2] Sheu, M.-H., and Lin, S.-H.: “Fast compensative design approach of the approximate squaring function”, IEEE J. Solid State Circuits, 2002, 37, (1), pp. 95–97.
- [3] Stallings, W.: “Cryptography and network security: principles and practice” (Prentice-Hall, 2003, 3rd edn.), ISBN: 0-13-091429-0.
- [4] S. Gao, N. Chabini, D. Al-Khalili and P. Langlois, “Optimized realizations of large integer multipliers and squarers using embedded blocks”, IET Computer Digital Technology, 2007.
- [5] Gao, S., Chabini, N., Al-Khalili, D., and Langlois, P.: “Optimized multipliers for large unsigned integers”. Proc. 23rd IEEE NORCHIP Conf., Oulu, Finland, November 2005, pp. 78–81.
- [6] Florent de Dinechin and Bogdan Pasca. “Large multipliers with less DSP blocks”. Technical Report 2009-03, LIP, École Normale Supérieure de Lyon, 2009.
- [7] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms* (MIT Press, 2000).
- [8] A. Karatsuba and Yu. Ofman (1962). "Multiplication of Many-Digital Numbers by Automatic Computers". *Proceedings of the USSR Academy of Sciences* 145: 293–294.
- [9] D. Strenski. FPGA floating point performance – a pencil and paper evaluation. HPCWire, Jan. 2007.