FPGA Implementation of the Multiplication Operation in Multiple-Precision Arithmetic

Kamil Rudnicki
Department of Reconfigurable Systems
Brightelligence Inc.
Glasgow, UK
Email: kamil.rudnicki@brightelligence.co.uk

Tomasz P. Stefański
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
80-233 Gdansk, Poland
Email: tomasz.stefanski@pg.gda.pl

***Abstract*—Although standard 32/64-bit arithmetic is sufficient to solve most of the scientific-computing problems, there are still problems that require higher numerical precision. Multiple-precision arithmetic (MPA) libraries are software tools for emulation of computations in a user-defined precision. However, availability of a reconfigurable cards based on field-programmable gate arrays (FPGAs) in computing systems allows one to implement MPA algorithms in hardware. Whereas addition and subtraction operations of two n-digit numbers require $O(n)$ operations, the basecase multiplication is equivalent to the convolution computation that requires $O(n^2)$ operations. Therefore, an efficient implementation of the multiplication operation is crucial for application of the reconfigurable hardware in MPA computations. In this contribution, our implementation of the basecase-multiplication algorithm in MPA on FPGA is presented. The method is implemented using the very high speed integrated circuit hardware description language (VHDL) and benchmarked on Xilinx Artix-7 FPGA. In the developed implementation of the MPA multiplication, the multiplication of two integer 1024-bit numbers (2048-bit numbers) takes 205 nsec (819 nsec) with the use of 40 DSP modules. It gives two-fold speedup in comparison to the reference results published in the literature. The developed digital circuit of the MPA multiplier works with integer numbers of precision varying in the range between 16 bits and 32 kbits. Such a scalability allows one to use the developed method not only in scientific computing, but also in embedded systems employing encryption based on MPA.**

***Keywords*—FPGAs, multiple-precision arithmetic, scientific computing, parallel processing, embedded systems.**

I. Introduction

Multiple-precision arithmetic (MPA) can be considered as a new approach for solving scientific problems which are difficult in terms of convergence, ill-conditioning and precision. It may be anticipated that MPA will be an increasingly important research tool in the future. The approach of a new era of computing is predicted in the literature [1], [2], in which the numerical precision required for computations is as important to the design of scientific codes as are algorithms and data structures. The review of MPA applications in scientific computing is presented in [1], [2]. Further MPA applications in computational electromagnetics are presented in [3]. Currently, MPA applications in scientific computing can be classified into the following categories:

- Generation of special mathematical functions applicable in the mathematical modeling.
- Solving ill-conditioned linear systems of equations.
- Difficult simulations of scientific and engineering problems (e.g., climate modeling, fluid dynamics, etc.).
- Derivation and verification of novel formulas and theorems in mathematics.

Elementary algebraic operations such as addition, subtraction and multiplication are crucial for any investigations requiring MPA computations. Whereas the addition and subtraction of two n-digit numbers require $O(n)$ operations, their multiplication requires $O(n^2)$ operations in the standard implementation (referred to as the basecase multiplication here). Therefore, the main effort in the implementation of the elementary MPA computations focuses on the multiplication operation. Although MPA multiplication can employ the Karatsuba, Toom-Cook and Schönhage-Strassen methods (which respectively require $O(n^{1.585})$, $O(n^{1.465})$, $O(n\log(n)\log(\log(n)))$ operations [4], [5], [6]), these algorithms are efficient for very-high-precision operands. For small precision of MPA operands, the extra shift and addition operations in those aforementioned algorithms make them slower than the basecase method. Therefore, for instance, GNU MPA library (GMP) [7] switches the multiplication method from the basecase to Karatsuba method when precision of operands is increased over 1920 bits (30 limbs, each limb equals 64 bits). Hence, efficient implementation of the basecase multiplication still remains a fundamental requirement for any MPA computations, regardless of the field of application, software tools and architecture of processing units.

There are several MPA libraries running on central processing units (CPUs), e.g., ARPREC, GMP, QD, MPIR (GMP-compatible) [3]. In the last few years, graphics processing units (GPUs) have been applied as a low-cost architecture for an acceleration of scientific codes. Therefore, MPA libraries were also implemented on GPU. There are several MPA libraries running under this parallel architecture, e.g., GARPEC, GQD, CUMP (GMP-compatible) [3]. However, the emulation of MPA computations on standard processors provides its significant slowdown when the precision of computations is increased. According to [2], computations on CPU in double-double precision typically run 5–10 times slower in comparison with 64-bit implementation. These slow down

at least 25 times for quad-double arithmetic, more than 100 times for 100-digit arithmetic, and over 1000 times for 1000-digit arithmetic. These facts demonstrate the need for new computing architectures accelerating MPA computations.

The popularity and promotion of a reconfigurable-computing hardware (e.g., refer to [8]) encouraged us to develop the implementation of the basecase-multiplication algorithm in MPA on a field-programmable gate array (FPGA). The method is implemented using the very high speed integrated circuit hardware description language (VHDL) and benchmarked on low-cost Xilinx Artix-7 FPGA [9]. Our results are compared with results reported in [10] demonstrating the advantages of the developed implementation. Due to application of the low-cost FPGA hardware in the research, the proposed multiplication method facilitates not only investigations in the area of scientific computing, but also can be useful for the embedded-system design with cryptographic solutions requiring MPA. For instance, modern encryption algorithms such as RSA and AES [11], [12], [13] employ keys whose length is higher than 256 bits. Currently, popular RSA keys are of length 1024 to 2048 bits, even though some experts believe that 1024-bit keys may become breakable in the near future [13]. Therefore, it can generally be assumed that RSA is secure if encryption keys are sufficiently long. As a result, critical systems employ dedicated cryptographic processors to protect information and to offload the related computational overhead (e.g., refer to [14]). To sum up, although the presented direction of the research is closely related to scientific computing, the results obtained might also be useful for development of electronic systems employing encryption.

II. Developed Implementation

The VHDL code for MPA multiplication is developed for Artix-7 FPGA using Vivado Design Suite [15]. The multiplication operation is executed on arrays of MPA numbers by the developed multiplier, which employs several multiplying units working in parallel. Each multiplying unit consists of a single DSP48E1 module available in this FPGA family [16]. The developed multiplier is scalable in terms of the number of multiplying units. It means that the multiplier can consist of a varying number of multiplying units, depending on the required processing power. In this contribution, we present the multiplication of two arrays of integer MPA numbers on a single multiplier consisting of a single multiplying unit for the sake of brevity. The multiplication operation for floating-point numbers can be obtained similarly, taking into consideration exponents and rounding, necessary to fit the output in fixed-precision (i.e., fixed-length) mantissa array of the result.

A. MPA Format of Integers

In the developed implementation, the format of integer numbers is the same as in the GMP library. It consists of an array of bytes representing the MPA number and its size. The absolute value of the size represents the length of the data array, whereas a sign of the size corresponds to a sign of the MPA number. Therefore, in the case of multiplication, the sign of the result can be easily computed as the multiplication of operand signs. The developed digital circuit of the MPA multiplication works with precision varying in the range between 16 bits and 32 kbits. Such a range is not a limiting step for MPA applications because the basecase multiplication for very-high-precision MPA operands is not efficient and it is worth using in this case, e.g., the Karatsuba method of multiplication on standard CPU.

B. Basecase Multiplication Algorithm

The basecase multiplication of two n-bit and m-bit numbers is a straightforward rectangular set of cross-products, the same as long multiplication done by hand, and for that reason sometimes known as the schoolbook or grammar school method [5], [7]. To some extent, the basecase multiplication of two numbers in any numerical system is equivalent to the multiplication of polynomials, which can be considered as a convolution computation. Hence, this is an $O(n \times m)$ algorithm.

C. Design Considerations

When designing a module in FPGA, one should take into account factors such as frequency of operation, resource utilization, congestion, overall performance and power efficiency. For computing applications, the overall performance is usually the most important factor, and consequently all considerations here are focused on achieving that purpose. Since the performance of the multiplication is directly proportional to the frequency of operation, this frequency should be kept at the maximum level. By doing so, the maximum distance to travel by a signal between two registers is shortened, so practically the design requires additional registers, which extend the range at the expense of latency. To sum up, the purpose of the design process is to find the trade off between the frequency of FPGA operation, and the latency.

D. Application of DSP Module

The DSP48E1 module available in Artix-7 FPGA enables computations of several arithmetic functions, which can be programmed into it. However, for the purpose of this project, only operation $\mathbf{A} \cdot \mathbf{B} + \mathbf{C} + \mathbf{CARRY\_IN}$ is employed. The maximum width of the operands is as follows:

- $\mathbf{A}$ - 25 bits
- $\mathbf{B}$ - 18 bits
- $\mathbf{C}$ - 48 bits
- $\mathbf{CARRY\_IN}$ - 1 bit.

Symmetric operands are used, due to the fact that asymmetric operand multiplier (25 $\times$ 18 bits) is difficult in construction and data storage (i.e., partial results are stored in an accumulator). Therefore, the maximum width of operands in the multiplying unit should be set at 18 $\times$ 18 bits. However, such a non-electronic number (i.e., different than power of 2) lowers the utilization of the BRAM memory block available in the FPGA chip. Consequently, the operand width is set to 16 $\times$ 16 with operand C of size 32 bits. For high performance,

such a configuration of the DSP module sets the latency at 4 clock cycles. In order to process the data out of, and into, the DSP module, a single additional clock cycle is required. Furthermore, the multiplexer for the data input requires a register in order to maintain the high speed of operation. This sets the minimum latency at 6 clock cycles, which is the final value of the minimum latency in the design process. Consequently, the number of channels put through the DSP module stems from the latency.

E. Implementation of Accumulator in BRAM

In order to multiply two numbers, an accumulator is required to store partial results. The BRAM memory [9] is used as the accumulator for the multiplication operation. Usually, the size of the accumulator ($2 \cdot prec$) is twice the precision of the input number ($prec$). This can be optimized to reach the accumulator size equal to $prec$. When the partial multiplication result is a part of the final result, such a result does not have to be stored in the accumulator any more, and can be passed to the output, e.g., refer to Table I. This way, when using one BRAM of 32 kbits, the precision of the output result can be up to 64 kbits. In the current implementation, the BRAM size determines the maximum precision of the input (not output) MPA numbers.

TABLE I
EXEMPLARY PARTIAL 16×16-BITS MULTIPLICATIONS AND SUMMATIONS IN THE ACCUMULATOR DEMONSTRATING THE UTILISATION OF THE FPGA RESOURCES

				4732	**9856**	**2397**	**5829**
			$\times$	**6796**	**5765**	**3454**	**3732**
			0F59	C69C	4278	7780	0702
		0E8D	B2E5	421A	AE09	B8F4	
	184E	5749	12F9	BBC6	6536		
1CCF	2B3E	4213	D21F	E2CC			

accumulator
output port

While the DSP module operates at 500 MHz (fast clock), the channel and BRAM memory do not run at such a high frequency. Hence, the frequency of each data unit (each multiplexed channel containing the BRAM memory) is set at 250 MHz (slow clock). The BRAM latency is equal to 2 clock cycles. A single extra clock cycle, needed for data processing, sets the total latency at 3 slow clock cycles, which corresponds well with 6 fast clock cycles.

F. Digital Circuit Design

Our digital circuit is designed to multiply numbers in streams, so there is no overhead needed for loading and off-loading data to and from the multiplier. Fig. 1 presents application of the DSP48E1 module available in Artix-7 FPGA in the designed digital circuit. As seen, the data processing requires time multiplexed channels in order to achieve full utilization of the multiplying unit. This is a consequence of the fact that the multiplier has its own latency, and that results of partial multiplications depend on the outcome of the previous ones. The number of channels depends on the latency introduced by the multiplying units and additional registers. Low latency multiplication can be achieved by elimination of extra registers for data transfer, which results from lack of congestion (equivalent to the low number of channels, which brings the design process to the starting point).

The basecase multiplication can be sped up by using a wider multiplying unit. Even though the multiplication latency is greater, and requires more additional registers to ease congestion, the overall performance is higher. For instance, by swapping the 16×16 bits multiplying unit for a 32×32 bits one, the overall performance increases by 33 % (with number of channels compensation included). What is more, the latency per each full number operation increases by 3 times. Furthermore, there is a computation-time quantization, which also increases 3 times. For an 18-channel multiplying unit, the multiplication of 19 pairs of numbers takes nearly double the time of the multiplication of 18 pairs of numbers. The more pairs of numbers, this extra time becomes more and more irrelevant, so a wider multiplier should be considered when large arrays of numbers are to be processed.

We investigate the performance of the MPA multiplication for varying precision and number of channels in the multiplier. The precision of the multiplying unit expressed as an electronic number (i.e., equal to the power of 2) results in maximum efficiency of the BRAM memory usage in the form of the accumulator. The speedup factor indicates that the 64-bit precision of the multiplying unit should be used. However, 34 channels create congestion in this case. Whereas each channel could be located far away from another, there is one focus point to which all channels must converge, i.e., the multiplying unit. Therefore, the number of channels should be kept to a minimum as this results in the highest possible frequency of operation. Keeping the number of channels to the minimum is also encouraged by the fact that it enables more uniform power distribution. That is, the application of smaller multipliers allows you to implement more of them in a single FPGA chip.

III. RESULTS

The VHDL code is synthesized and implemented for Xilinx Artix-7 FPGA. Its correctness is verified at 300 MHz with the use of the Nexys Video board from Digilent [17]. Here, we present performance evaluation for Artix-7 implementation at frequency of 500 MHz with multiplication size 16×16 bits and 6 channels per DSP module. These are pure execution times of the multiplication operation. Hence, presented results do not take into account data transfer through PCI-E bridge and DDR4 memory, which will introduce additional latency. Results for a varying number of DSP modules, compared to the reference results in [10], are presented in Table II. The reference results are obtained for Virtex-6 FPGA [18], which is also equipped with DSP48E1 slices. Hence, such a comparison is justified by the application of the same DSP module.

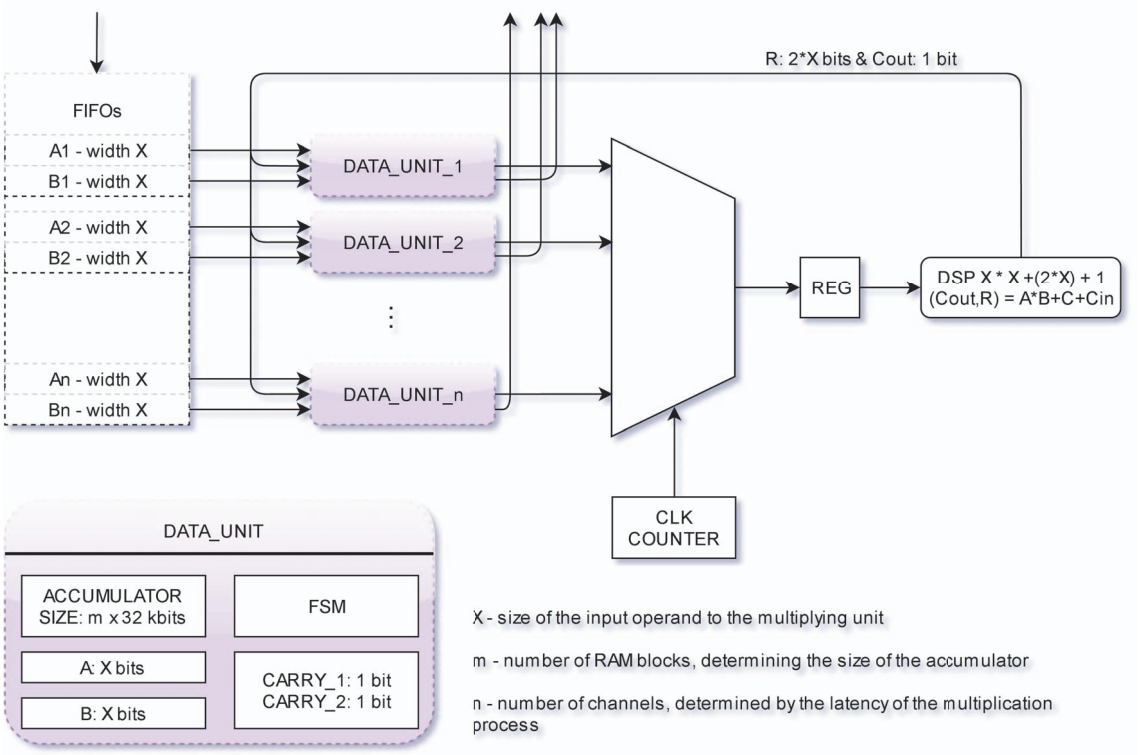

Fig. 1. Application of the DSP48E1 module in Artix-7 FPGA in the digital circuit implementing the MPA multiplication

TABLE II
EXEMPLARY EXECUTION TIMES OF MPA MULTIPLICATION IN COMPARISON WITH RESULTS IN [10]

precision (bits)	**[10]** (μsec)	**Artix-7 @ 500 MHz** (μsec)		
		5 DSP	20 DSP	40 DSP
1024	0.41	1.6	0.41	0.205
2048	1.30	6.6	1.6	0.819

In our test, arrays of 1, 000, 000 random MPA numbers are multiplied, and the execution time of a single multiplication is calculated for a single output number. As seen, the developed solution outperforms reference results [10] for a sufficient number of DSP modules employed in the MPA-multiplication operation. Furthermore, the execution times decrease almost linearly with the number of DSP modules employed for the MPA multiplication.

For the basecase multiplication, an equation can be derived to calculate the average time needed to multiply two numbers, i.e.

$$t_{per\_number} = \left(\frac{prec}{mult\_prec}\right)^2 \cdot \frac{1}{mult\_units \cdot freq} \quad (1)$$

$$t_{measured} = t_{per\_number} \cdot table\_size \quad (2)$$

where t_{per_number} is the average time for calculation of a single MPA-multiplication result, $prec$ is the precision of the numbers to be multiplied, $mult\_prec$ is the precision of the multiplying unit, $table\_size$ is the number of multiplications, $mult\_units$ is the number of multiplying units working in parallel performing multiplication operations, $freq$ is the frequency of operation of DSP modules and $t_{measured}$ is the time needed to multiply two arrays.

There is also an additional dependency between $mult\_prec$, $mult\_units$, $freq$ and $table\_size$. The rule of thumb dictates that the higher the $mult\_prec$, the lower the maximum number of multiplying units available in a single FPGA chip. This is due to the increase of latency for DSP modules, that forces an increase in the number of channels, which results in a higher resource utilization. Such an increase in the resource utilization may also limit the maximum speed of operation ($freq$). The higher the number of channels, the higher $table\_size$ is required in order to limit the computation-time quantization error.

Our design efficiency per number of DSP modules is over two times higher than for the processor presented in [10]. This is mainly due to the fact that our design runs at two times higher frequency (500 MHz). Furthermore, in order to run 64-bit operations on DSP modules, the individual modules have to exchange data, which requires additional stage registers and this lowers the efficiency. The DSP modules in our solution work independently, hence, the higher efficiency. However, such a comparison has limited validity because consumption of other resources should also be taken into account.

Application of the proposed solution requires sending a data stream to the (e.g., memory-mapped) multiplier, whereas the solution [10] is based on a finite-state machine, which executes the multiplication operation depending on a micro-instruction set. Both solutions represent completely different approaches to the MPA multiplication and it limits the possibility of fair comparison.

The solution proposed here offers a fine scalability, and allows you to run MPA multiplication, even on the low-cost FPGA chips available currently on the market, i.e., any Artix, most of Spartan 7, and almost all Spartan 6 family chips. Being runnable on any Zynq based platform, our solution is suitable for embedded applications requiring the MPA multiplication. Such a need may stem from, e.g., an application of a demanding encryption method in a communication.

IV. Conclusion

Currently, there are still scientific problems that require higher numerical precision than provided by the standard 32/64-bit arithmetic. Availability of reconfigurable cards based on FPGAs in computing systems, allows one to implement MPA algorithms in hardware. In this contribution, our implementation of the basecase-multiplication algorithm in MPA on Xilinx Artix-7 FPGA is presented. In the developed implementation of the MPA multiplication, the multiplication of two integer 1024-bit numbers (2048-bit numbers) takes 205 nsec (819 nsec) with the use of 40 DSP modules. It gives an almost two-fold speedup in comparison to the reference results published in the literature. The developed digital circuit of the MPA multiplier works with integer numbers of precision varying in the range between 16 bits and 32 kbits with a step size of the multiplying unit precision (16 bits). Therefore, the results are hopefully useful not only in scientific computing, but can also be useful for design of embedded systems with cryptographic solutions requiring MPA.

Acknowledgment

Tomasz Stefański is grateful to Cathal McCabe at Xilinx Inc. for arranging the donation of design software tools and Wojciech Żebrowski at Aldec Inc. for his support. This work was supported in part under funding for Statutory Activities for the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology.

References

[1] D. H. Bailey, "High-precision floating-point arithmetic in scientific computation," *Comput. Sci. Eng.*, vol. 7, no. 3, pp. 54–61, 2005.

[2] D. H. Bailey, R. Barrio and J. M. Borwein, "High-precision computation: Mathematical physics and dynamics," *Appl. Math. Comput.*, vol. 218, no. 20, pp. 10106–10121, 2012.

[3] T. P. Stefanski, "Electromagnetic problems requiring high-precision computations," *IEEE Antennas Propag. Mag.*, vol. 55, no. 2, pp. 344–353, 2013.

[4] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Proceedings of the USSR Academy of Sciences*, vol. 145, pp. 293–294, 1962.

[5] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[6] A. Schönhage and V. Strassen, "Schnelle multiplikation grosser zahlen," *Computing*, vol. 7, no. 3, pp. 281–292, 1971.

[7] T. Granlund, "The GNU multiple precision arithmetic library (Edition 6.1.2)," GMP Development Team, 2016 [Online]. Available: www.gmplib.org

[8] SDAccel Platform Reference Design User Guide - Developer Board for Acceleration with KU115, UG1234 (v2016.3) November 30, 2016 [Online]. Available: www.xilinx.com

[9] 7 Series FPGAs Data Sheet: Overview - Product Specification, DS180 (v2.2) December 15, 2016 [Online]. Available: www.xilinx.com

[10] Y. Lei, Y. Dou, S. Guo, J. Zhou, "FPGA implementation of variable-precision floating-point arithmetic," in: *O. Temam, PC. Yew, B. Zang (eds) Advanced Parallel Processing Technologies. APPT 2011. Lecture Notes in Computer Science*, vol. 6965, Springer, Berlin, Heidelberg, 2011.

[11] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[12] J. Daemen, V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*., Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[13] A. Kalathungal, *An Arbitrary Precision Integer Arithmetic Library for FPGAs*., MSc thesis, University of Cincinnati, OH, USA, 2013.

[14] IBM, System z9 Enterprise Class, System Overview, April 2009 [Online]. Available: www-01.ibm.com

[15] Vivado Design Suite User Guide - Getting Started, UG910 (v2016.3) October 5, 2016 [Online]. Available: www.xilinx.com

[16] 7 Series DSP48E1 Slice User Guide, UG479 (v1.9) September 27, 2016 [Online]. Available: www.xilinx.com

[17] Nexys Video FPGA Board Reference Manual, Revised May 4, 2016 [Online]. Available: www.digilentinc.com

[18] Virtex-6 Family Overview - Product Specification, DS150 (v2.5) August 20, 2015 [Online]. Available: www.xilinx.com