Proceedings of the 25th International Conference *"Mixed Design of Integrated Circuits and Systems"*
June 21-23, 2018, Gdynia, Poland

IP Core of Coprocessor for Multiple-Precision-Arithmetic Computations

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.edu.pl

***Abstract*—In this paper, we present an IP core of coprocessor supporting computations requiring integer multiple-precision arithmetic (MPA). Whilst standard 32/64-bit arithmetic is sufficient to solve many computing problems, there are still applications that require higher numerical precision. Hence, the purpose of the developed coprocessor is to support and offload central processing unit (CPU) in such computations. The developed digital circuit of the coprocessor works with integer numbers of precision approaching maximally 32 kbits. Our IP core is developed using the very high speed integrated circuit hardware description language (VHDL) and simulated assuming implementation in field-programmable gate arrays (FPGAs). It exchanges data using three 64-bit data buses whereas a code for execution on the coprocessor is fetched from a dedicated 8-bit bus (all buses in AMBA standard - AXI Stream). An instruction set of the coprocessor currently consists of 7 instructions including multiplication, addition and subtraction. The computations can maximally employ 16 registers of the length 32k bits. Simulation results assuming implementation on Zynq system on chip (SoC) show that computations of the factorial ($n!$) for n=1000 take 326.4 μsec. Such a design currently requires 7982 look-up tables (LUTs), 10400 flip-flops (FFs), 33 block RAMs (BRAMs) and 28 DSP modules. The processor is aimed to provide scalability allowing one to use the developed IP core 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

The increasing importance of scientific and engineering computations in arithmetic higher than the standard 32/64 bits is the motivation for our research. Although the standard 32/64-bit arithmetic is sufficient to solve many scientific problems, a broad range of computations still cannot be executed with such precision. Multiple-precision arithmetic (MPA) is a computational tool that allows one solving numerically difficult problems. MPA can be considered as a new approach to tackle open problems in science that remain unsolved due to the complexity of computations. The advent of a new era of scientific computing is predicted in the literature [1], [2], in which the numerical precision required for computations is as important to the code design as are the algorithms and data structures. Therefore, MPA has already found applications in scientific computing which include: (i) Generation of special mathematical functions applicable in scientific computations (e.g., discrete Green's function, Bessel functions, etc.). (ii) Solving ill-conditioned linear systems of equations. (iii) Derivation and verification of novel formulas and theorems in mathematics. (iv) Difficult simulations of processes with limited stability (e.g., climate modeling, fluid dynamics, analysis of radio-frequency large-scale circuits, electrodynamics). (v) Cryptography, cryptanalysis, number theory. The general review of MPA applications in scientific computing is published in [1], [2]. The review of MPA applications in computational electrodynamics is published in [3].

Unfortunately, MPA computations require large resources in terms of processor time and memory consumption. According to [2], computations on central processing units (CPUs) in double-double precision typically run 5-10 times slower than those implemented in 64-bit arithmetic. The slowdown is at least 25 times for quad-double arithmetic, more than 100 times for 100-digit arithmetic, and over 1000 times for 1000-digit arithmetic. Hence, the development of an MPA accelerator is needed, due to the overhead of MPA computations. Furthermore, such an accelerator should be scalable allowing one to execute arithmetic operations not only on single operands but also on arrays of data in parallel.

Whilst several MPA processor/coprocessor concepts have arisen in recent years [4], [5], [6], [7], [8], none of them have achieved commercial success and popularity. As a result, none of these architectures is available as a parallel MPA coprocessor useful for scientific computing or embedded systems design. Hence, we decided to develop from scratch an IP core of the MPA coprocessor with scalability enabling its implementation in the field-programmable gate arrays (FPGAs) of various scale. Although the presented core is currently investigated assuming an implementation in system on chip (SoC) devices (i.e. Zynq devices from Xilinx [9], [10]), it can further be implemented in FPGA PCIe accelerator cards. Therefore, the developed IP core should be applicable in scientific computing as well as in embedded systems with e.g. cryptographic solutions requiring MPA.

In this paper, we present the simulation results for a single IP core of the MPA coprocessor. It exchanges data with host CPU using three 64-bit data buses. Two of them deliver data to the coprocessor and one returns results of computations to CPU. A code for execution on the coprocessor is fetched from

a dedicated 8-bit bus. All these buses are in AMBA standard (AXI Stream) [11]. An instruction set of the coprocessor currently consists of 7 instructions including multiplication, addition and subtraction. These algebraic operations are crucial for any investigations requiring MPA computations, regardless of the field of application. In our design, computational overhead required to execute multiplication and addition/subtraction on two n-bit operands grows respectively as $O(n^2)$ and $O(n)$. The proposed coprocessor currently employs the standard implementation of multiplication only (referred to as the basecase multiplication here). Although the 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 [12], [13], [14]), 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. Hence, for assumed precision of MPA computations below 32 kbits, implementation of multiplication with the use of the basecase algorithm is sufficient.

Simulation results presented in the paper focus on factorial computations involving multiplications, which return results growing very fast for increasing values of input arguments. Due to the computational overhead of the multiplication, this operation is the limiting step of the proposed MPA coprocessor in such a benchmark.

II. Developed Coprocessor

The VHDL code of the MPA coprocessor is currently developed assuming implementation in Xilinx Zynq-7000 AP SoC XC7Z020-1CLG484C on Zedboard [9], [10], which includes FPGA and two 32-bit Arm A9 CPU cores. Vivado Design Suite [15] is employed as a software tool for this purpose. The design assumption is that the developed coprocessors should be scalable and able to work in parallel, depending on the required processing power and hardware available. Hence, the coprocessor is a parameterizable IP core, which can be implemented in different configurations, depending on the final application, in 7 series of Xilinx FPGAs [16]. In this paper, we present the operation of a single MPA coprocessor for the sake of brevity.

A. Architecture

The architecture of the developed MPA coprocessor is presented in Fig. 1. Instructions are fetched from the program bus to the instruction decoder. Then, control lines (Ctrl) are set according to a fetched instruction. For data buses A and B, loaders deliver data to the bank of 16 registers. Each register can store an MPA number of the maximal length 32 kbits. In the developed MPA coprocessor, the sign-magnitude representation is employed for integer numbers. Hence, each number consists of an array of bytes representing the number magnitude (with limb size equal to 64 bits), a sign bit and size bits. Therefore, in the case of multiplication, the sign of the result can be easily computed as the multiplication of operand signs. Numbers are directed to arithmetic units (i.e., multiplier, adder/subtractor) depending on multiplexer settings stemming from an instruction executed. Results of arithmetic operations are stored back in the bank of registers. Finally, results of computations can be transferred into CPU using the module of unloader. The number of data buses and registers can be varied for the coprocessor as a parameter of the IP core, depending on hardware specification and final application.

B. Instruction Set

Currently, the instruction set consists of 7 instructions, which are summarized in Table I. Instructions **loaa**, **loab**, **loaab** allow one to set data in registers from data buses A and B. Instruction **unl** allows one to transfer result of computations from a register to host CPU using output data bus. Instructions **mult**, **add**, **sub** are respectively arithmetic operations of multiplication, addition and subtraction. Although the number of instructions is currently limited, essential computations requiring MPA can be executed and the evaluation of the coprocessor performance can be carried out.

TABLE I
Instruction set of the developed MPA coprocessor

Instruction	Result
loaa regX	regX = data(busA)
loab regX	regX = data(busB)
loaab regX, regY	regX = data(busA) and regY = data(busB)
unl regX	data(busO) = regX
mult regX, regY, regZ	regZ = regX * regY
add regX, regY, regZ	regZ = regX + regY
sub regX, regY, regZ	regZ = regX - regY

regX, regY, regZ denote any of 16 registers of length 32 kbits
busA, busB, busO denote respectively bus A, bus B and output bus.

C. Digital Circuit Design

Currently, the developed circuit of the MPA coprocessor consumes the following resources: look-up tables (LUTs): 7982, flip-flops (FFs) : 10400, block RAMs (BRAMs): 33, DSP modules: 28. The developed circuit can currently work with the clock frequency maximally equal to 400 MHz using Vivado Synthesis Defaults and Vivado Implementation Defaults. Our future work is to optimize the design and increase this frequency up to around 500 MHz.

III. Simulation Results

The developed MPA coprocessor is benchmarked in factorial computations. For this purpose a personal computer (PC) code, which generates input data and instructions for such computations, has been developed. Furthermore, this code allows one to generate not only binary data for the MPA coprocessor, but also assembler and simulator input files.

Listing of the code for factorial computations is presented in Table II for n=4. In lines 1 and 2, reg0, reg2, reg3 are loaded with initial data equal to 1. Then (line 3), value of reg4 is obtained by adding reg2 and reg3 (reg4=2). In the next line (line 4), reg4 is multiplied by reg0 and result is stored in reg1 (reg1=2). Then (line 5), value of reg2 is obtained by adding reg4 and reg3 (reg2=3). In the next line (line 6), reg2

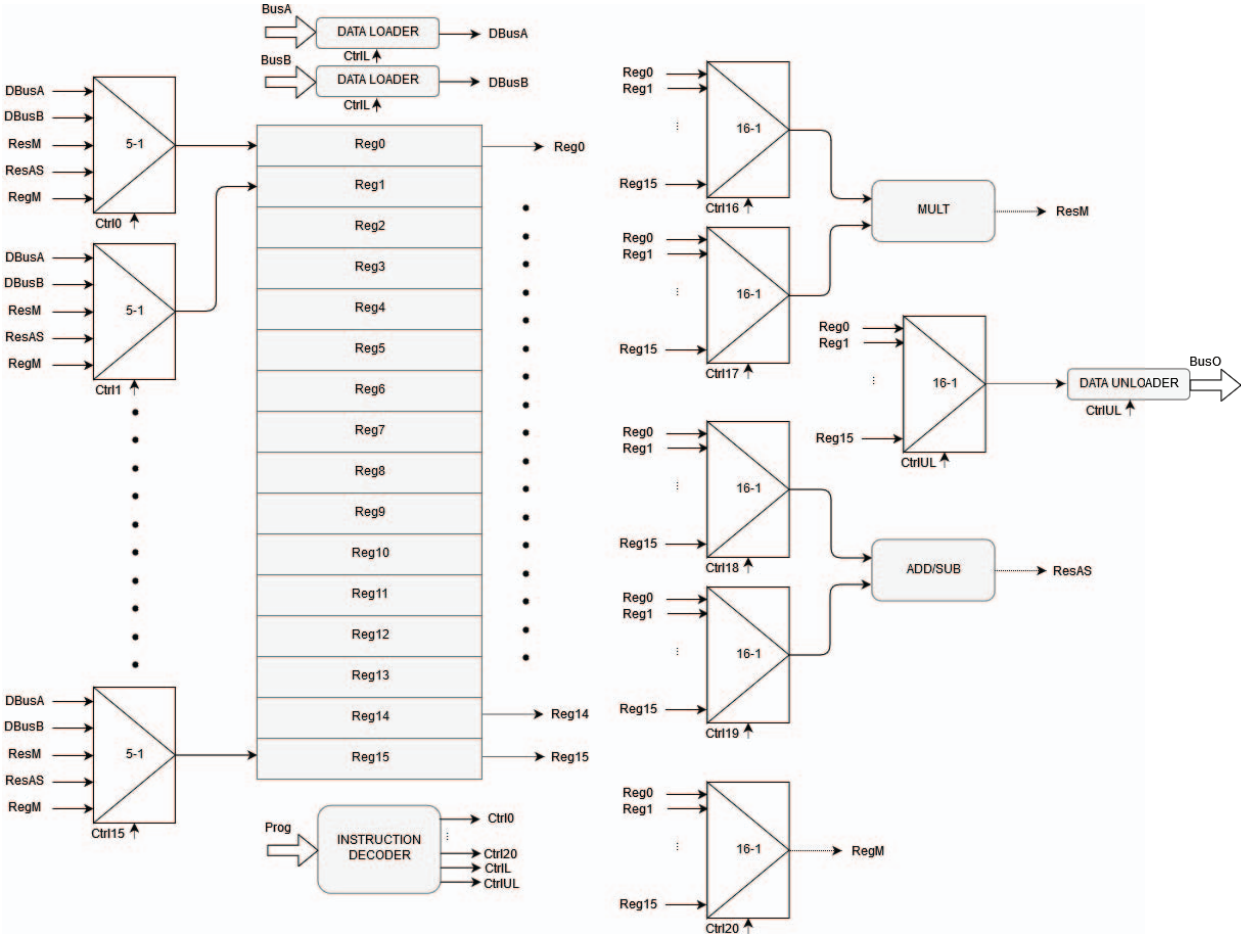

Fig. 1. Architecture of the MPA coprocessor.

is multiplied by reg1 and result is stored in reg0 (reg0=6). Then (line 7), value of reg4 is obtained by adding reg2 and reg3 (reg4=4). In the next line (line 8), reg4 is multiplied by reg0 and result is stored in reg1 (reg1=24). Finally, value of reg1 is unloaded in line 9.

TABLE II
LISTING OF CODE FOR FACTORIAL COMPUTATIONS (n=4)

Line	Instruction
1	**loaab** reg0, reg2;
2	**loaa** reg3;
3	**add** reg2, reg3, reg4;
4	**mult** reg4, reg0, reg1;
5	**add** reg4, reg3, reg2;
6	**mult** reg2, reg1, reg0;
7	**add** reg2, reg3, reg4;
8	**mult** reg4, reg0, reg1;
9	**unl** reg1;

reg0, reg2, reg3 are loaded with 1.

The correctness of computations is confirmed in Fig. 2, where waveforms for the simulation of 4! computations are presented. As seen, the result in reg1 at time 646 nsec is equal to 24 (0x18).

For the sake of comparison, the reference code has been developed with the use of the GMP library [17], which allows one to compute factorials on linux-based PCs. In Fig. 3, measured times of factorial computations ($n!$) are presented for n varying in the range up to n=1000. These computations are executed on single cores of Intel i7 (i7-7700 CPU @ 3.60GHz) and Arm Cortex A9 (Xilinx Zynq-7000 AP SoC XC7Z020-1CLG484C on Zedboard @ 667MHz) processors. Performance of the developed MPA coprocessor is estimated based on simulation results for factorial computations obtained in simulation software [15]. For n=1000, a single core of i7 (A9) processor needs 272 (893) μsec whereas the MPA coprocessor (FPGA) needs 326.4 μsec according to obtained simulation results. Hence, the developed coprocessor should be only 1.2 times slower than a core of modern i7 processor and 2.7 times faster than a core of A9 processor within SoC. It shows that the embedding of MPA computations in programmable logic within SoC may provide the speedup of these computations. Furthermore, it is worth noticing that with the use of resources available on Zynq-7000 SoC, the implementation of four MPA cores is possible in hardware. Hence, the proposed MPA coprocessor can be a valuable solution for offloading the MPA computations from CPU.

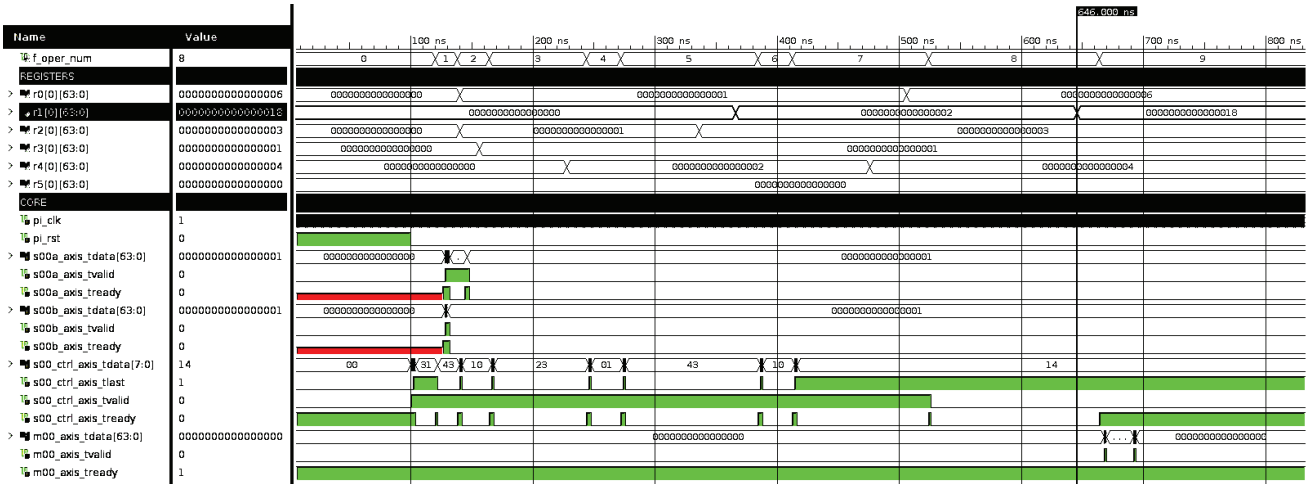

Fig. 2. Simulation waveforms for 4! computations.

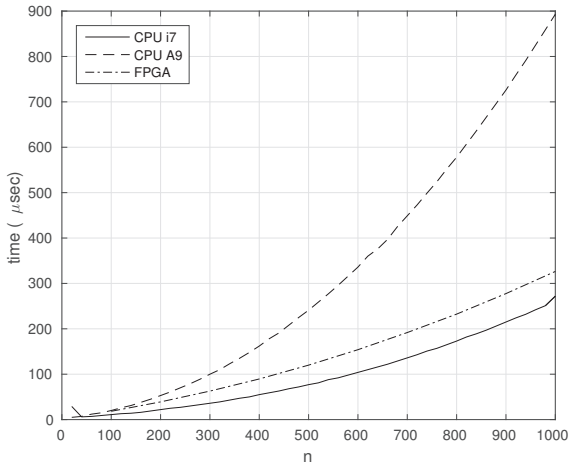

Fig. 3. Times of factorial computations ($n!$). Results for CPUs are measured whereas FPGA performance is estimated from simulations.

IV. Conclusion

The IP core of coprocessor supporting computations requiring integer MPA is developed. Currently, its performance in MPA factorial computations is estimated based on simulation results. The results obtained demonstrate that such a coprocessor can accelerate 2.7 times MPA computations within SoC in comparison to GMP computations on 32-bit CPU core. The IP core of the developed coprocessor is aimed to provide scalability allowing one to use it in scientific computing as well as embedded systems employing encryption based on 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. Nadjia, A. Mohamed, B. Hamid, I. Mohamed and M. Khadidja, "Hardware algorithm for variable precision multiplication on FPGA," *2009 IEEE/ACS International Conference on Computer Systems and Applications*, Rabat, pp. 845-848, 2009.

[5] A. F. Tenca and M. D. Ercegovac, "A variable long-precision arithmetic unit design for reconfigurable coprocessor architectures," *1998 IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, pp. 216-225, 1998.

[6] Y. Lei, Y. Dou, S. Guo, and J. Zhou, "FPGA Implementation of Variable-Precision Floating-Point Arithmetic," *9th International Symposium, APPT 2011*, Shanghai, China, pp. 127-141, 2011.

[7] J. Hormigo, J. Villalba, E. L. Zapata, "Cordic processor for variable-precision interval arithmetic," *Journal of VLSI Signal Processing*, vol. 37, no. 1, pp. 21-39, 2004.

[8] M. J. Schulte, E. E. Swartzlander Jr., "A family of variable-precision, interval arithmetic processors," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 387-397, 2000.

[9] Zynq-7000 All Programmable SoC Data Sheet: Overview - Product Specification, DS190 (v1.11) June 7, 2017 [Online]. Available: www.xilinx.com

[10] ZedBoard (Zynq Evaluation and Development) - Hardware User's Guide, Version 1.1, August 1st, 2012 [Online]. Available: www.digilentinc.com

[11] Vivado Design Suite - AXI Reference Guide, UG1037 (v4.0) July 15, 2017 [Online]. Available: www.xilinx.com

[12] 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.

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

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

[15] Vivado Design Suite User Guide - Getting Started, UG910 (v2017.3) October 4, 2017 [Online]. Available: www.xilinx.com

[16] 7 Series FPGAs Data Sheet: Overview - Product Specification, DS180 (v2.6) February 27, 2018 [Online]. Available: www.xilinx.com

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