Marcin PIETROŃ, Kazimierz WIATR
AKADEMIA GÓRNICZO-HUTNICZA, ACK – CYFRONET, ul. Nawojki 11, 30-950 Kraków
AKADEMIA GÓRNICZO-HUTNICZA, KATEDRA ELEKTRONIKI, Al. Mickiewicza 30, 30-059 Kraków

Elementary functions in HLL on example of CORDIC algorithm implemented in Mitrion-C language

Mgr inż. Marcin PIETROŃ

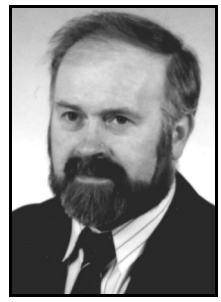
MS degree in Electronics and Telecommunication Engineering (2003), Computer Science (2005). Currently he is working toward doctorate degree in Computer Science at the Department of Electrical Engineering and Computer Science at the University of Science and Technology in Cracow. His research interests lie in hardwaresoftware co-design, high performance computing.

e-mail: pietron@agh.edu.pl

Prof. dr hab. inż. Kazimierz WIATR

MS degree in Electronics Engineering from University of Science and Technology, Cracow (1980), Ph.D. degree in Electronics Engineeering (1987), Professor of Electronics Engineering (2002). Currently he is professor with the Department of Electrical Engineering and Computer Science of University of Science and Technology in Cracow and Director of ACK Cyfronet AGH. His research interests focus on image processing systems, multi-processor systems and FPGA-based accelerator design.

e-mail: wiatr@agh.edu.pl



Abstract

The elementary functions are very often used in scientific computations. The quantum chemistry, physics, financial computing are only examples were elementary functions like exponent, logarithm are intensively computed. This paper presents implementation of an *exp(x)* core in a CORDIC-algorithm written in Mitrion-C lanuage. The Mitrion-C language is a new high level language. It enables implementing pipelined and wide paralleled algorithms on FPGA platforms. It makes process of algorithms implementation on FPGA faster. From gravitational forces to quantum chemistry or financial mathematics, computational scientists very often use *exp(x)* in computer simulations. The implemented core generates IEEE 754 standard single precision exponential values. The CORDIC algorithm can be used to compute wide spectrum of different elementary functions like sine, cosine, tangent. In our solution values of the exponent for integer part of the input argument are stored in a table. The table is allocated in an internal memory. The fractional part is computed by the CORDIC algorithm. The final result is achieved by multiplying the values of the fractional and integer part. Our implementation is made on SGI Altix 4700 hardware platform. It is SGI multiprocessor distributed shared memory computer system with Virtex-4 LX 200 FPGAs.

Keywords: HLL, FPGA, elementary functions, HPRC.

Implementacja funkcji elementarnych w FPGA na przykładzie algorytmu CORDIC w języku wysokiego poziomu Mitrion-C

Streszczenie

Funkcje elementarne są bardzo często wykorzystywane w obliczeniach naukowych. Chemia kwantowa, matematyka finansowa, fizyka jedne z wielu dziedzin gdzie funkcje takie jak eksponenta, logarytm są intensywnie wykonywane. Praca ta przedstawia implementację funkcji eksponenty za pomocą algorytmu CORDIC w języku Mitrion-C. Mitrion-C jest nowym językiem wysokiego poziomu programowania układów FPGA. Język ten posiada odpowiednie instrukcje oraz wbudowane typy danych, które pozwalają na programowanie algorytmów potokowo jak i całkowicie równolegle. W naszym rozwiązaniu argument wejściowy jest rozdzielony na część całkowitą i część ułamkową. Wartości eksponenty dla części całkowitej przechowywane są w tablicy w pamięci wewnętrznej natomiast część wartość dla części ułamkowej obliczana jest algorytmem CORDIC. Wynik końcowy obliczany jest za pomocą mnożenia części ułamkowej i całkowitej. Implementacja wykonana jest na platformie sprzętowej SGI ALTIX 4700. Jest to platforma wieloprocesorowa ze współdzieloną pamięcią oraz układami FPGA typu Virtex-4 LX 200.

Słowa kluczowe: HLL, FPGA, funkcje elementarne, HPRC.

1. Introduction

When implementing HPC computations on modern hardware architectures like HPRC (High Performance Reconfigurable Computing) time and efficiency of implementation are important issues.

Apart from programming FPGA in HDL like VHDL there are high level languages created (HLL) to make process of designing programmable circuits faster. They are built to support building complex algorithms on FPGA hardware platforms. They support top-down methodology of designing FPGA circuits. A programmer by using HLL can quickly estimate if an algorithm can be efficiently implemented in FPGA. Then it is possible to choose which parts of the algorithm can be implemented in HDL to achieve further speedup or to decrease area usage of programmable circuits. The Mitrion-C language is one of the most popular HLL in the last years in HPRC market.

In HPC scientific applications elementary functions like sine, cosine, logarithm, exponent etc. are very often massively computed. In this paper we use this language to implement exponential function in CORDIC algorithm and compare it with implementations of exponent in VHDL. The CORDIC algorithm is very popular for computing elementary functions using only addition, subtraction, bitshift and lookup table [5]. The research is performed on Altix 4700 SGI hardware platform.

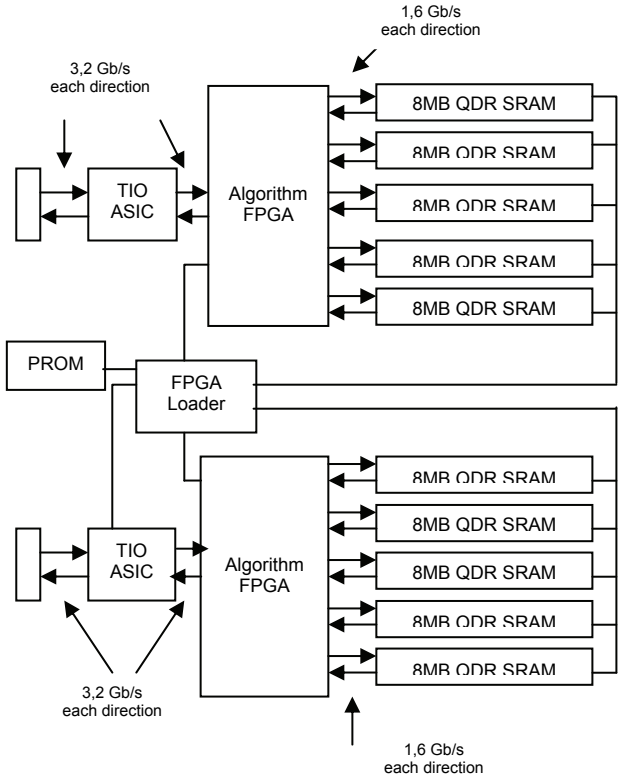
2. The SGI Altix 4700 platform

The Altix 4700 series is a family of the multiprocessor distributed shared memory computer systems, and it currently ranges from 8 to 512 CPU sockets (Fig. 1) [7]. Each processor has its own local memory as well as the ability to access very fast memories of other processors by a NUMALink connection. NUMALink allows dataflow of 6,4GB/s. SGI RASC RC100 Blade consists of two Virtex-4 LX 200 FPGAs, with 40MB of SRAM logically organized as two 16MB blocks and an 8MB block.

Each QDR SRAM block transfers 128-bit data every clock cycle (at 200MHz) both for reading and writing. The RASC communication module is based on application of a specific integrated circuit (ASIC) TIO, which attaches to the Altix system NUMAlink interconnect directly. TIO supports the Scalable System Port (SSP) that is used to connect the Field Programmable Gate Array (FPGA) with the rest of the Altix system. The RC100 Blade is connected using the low latency NUMALink interconnect to the SGI Altix 4700 Host System. NUMALink enables a bandwidth of 3.2GB per second in each direction. The Altix 4700 has its own built in development platform which gives the RASC API the ability to write programs on a host processor that invoke compiled VHDL source codes on the FPGA circuits. The second possibility of the development of the HPC application is to use the RC100 Blade to write source code in HLL. Mitrion-C is HLL which enables this process.

3. The Mitrion-C language

The most popular HLL on SGI Altix 4700 platform is Mitrion-C. The Mitrion-C compiler generates VHDL code from the Mitrion-C source. The design is then synthesized using the Xilinx suite of synthesis and implementation tools. Mitrion SDK consists of a Mitrion-C language compiler, an integrated development environment, a data dependency graph visualization and simulation tool, a Mitrion Host Abstraction Layer (MITHAL) library, and the target platform-specific processor configurator (Fig. 2).



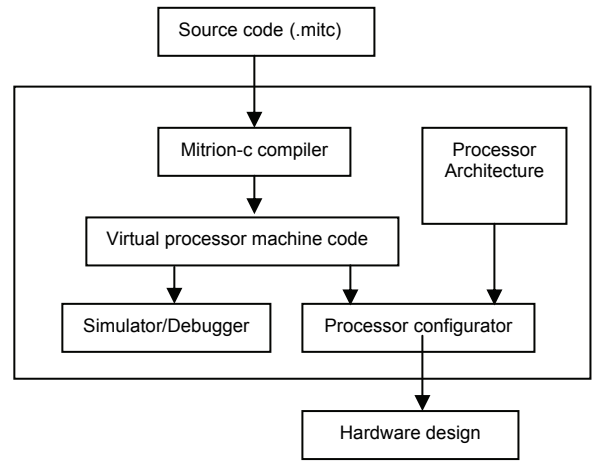
Rys. 1. Platforma sprzętowa SGI Altix 4700
Fig. 1. SGI Altix 4700 architecture

The Mitrion-C source code is compiled into an intermediate virtual processor machine code. This machine code can be used by a simulator, debugger or processed by a processor configurator to produce a VHDL design for the hardware platform. The Mitrion-C programming language is an intrinsically parallel language. The Mitrion-C data types, such as vectors, lists and language constructs, such as loops, are designed to support the parallel execution driven by the data dependencies. The main mechanisms of parallelism are *foreach* and *for* loop constructs. By using them we can achieve unrolled or pipelined code execution in FPGA (Tab.1) [7]. It also enables using all QDR SRAM memory banks. The parallelism and pipelining is achieved because Mitrion Virtual Processor is a fine-grain, massively parallel, reconfigurable processor. It is possible to specify thousands of simultaneously executing processes. The scheduling system ensures that data is at the correct place at the correct time. The Mitrion-C makes it impossible to create deadlocks. It prevents difficult debugging of large-scale parallel programs.

Tab. 1. Zależności pomiędzy pętlami i strukturami danych w języku Mitrion-C
Tab. 1. Loops and data structure dependency in Mitrion-C language

	Vector	List
foreach	wide parallel	pipelined
for	unrolled	sequential

Additionally, Mitrion-C has a special API that can be used to invoke Mitrion-C bitstream from C or Fortran language (the Mitrion-C Host Application Layer). It enables insertion of the Mitrion-C bitstream into existing C/C++ and Fortran codes.



Rys.2. Etapy programowania w języku Mitrion-C
Fig. 2. Mitrion-C development process

The syntax of the Mitrion-C is similar to C-family languages, like C, C++, Java, etc.

4. CORDIC algorithm

Very often microprocessors do not provide an exponential unit in hardware. Instead, that operation is implemented in software by combination of look-up tables and polynomial approximation. The floating-point CORDIC algorithm [2] is one in which a floating-point vector is rotated over a floating-point angle. The angle derived depends on whether the mode of operation is vectoring or rotation.

Our work concentrates on the rotation mode of CORDIC. The CORDIC method is based on a set of three equations, the angle z, x and the y coordinate.The x and y coordinate correspond to *cosh* and *sinh*, respectively. The difference of *cosh(x)* and *sinh(x)* for $x > 0$ is equal *exp(x)*. The CORDIC equations for hyperbolic rotations are shown below:

$$x_{i+1} = x_i + y_i * d_i * 2^{-i}, \tag{1}$$

$$y_{i+1} = y_i + x_i * d_i * 2^{-i}, \tag{2}$$

$$z_{i+1} = z_i - d_i * \tanh^{-1}(2^{-i}), \tag{3}$$

$$d_i = \begin{cases} -1 \rightarrow z_i < 0 \\ +1 \rightarrow otherwise \end{cases}, \tag{4}$$

The value of d_i is the direction of rotation; x and y are the vector elements. The CORDIC algorithm iterates to either zero y or zero z. The first angle to rotate is 45^o. Then angles in next steps are chosen in a way that tangent values are equal 2^{-i}.

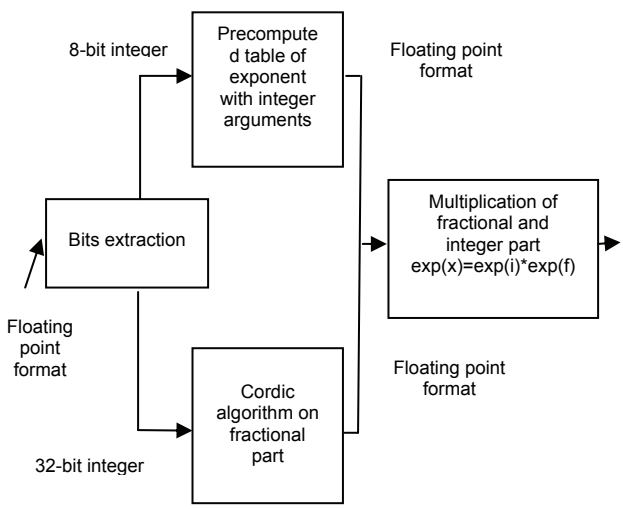
Eq. 5 shows how the vectors are rotated and dependencies between angles and coordinates of the vectors in each iteration.

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \begin{bmatrix} \cosh\Theta & \sinh\Theta \\ \sinh\Theta & \cosh\Theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}, \tag{5}$$

5. Implementation and results

In our implementation a little modification of the original algorithm is made. An input argument in the floating point format is divided into fractional and integer part ($x = i + f$). The values of the exponent of integer arguments are precomputed and stored in an internal memory. Eight bits of the integer part are extracted by a bit mask from the floating point number (eight bits are enough to represent all possible integers of exponent arguments in the floating point format). After extracting eight most significant bits, 32 bit integers are cast to eight bit format integer. The exponent of the fractional part is computed by CORDIC. The least 24 significant bits (fractional part) are extracted by the bit mask. The fractional part is widen to 32 bit fixed point format where two most significant bits are allocated for the integer part of CORDIC computation and the rest 30 bits are the fractional part. The body of our CORDIC algorithm uses *for* Mitrion-C loop to compute the exponent of the fractional part. It uses values from *ktab* and *htab* in each iteration [5]. The integer and fractional parts are computed independently (Fig. 3). At the end these two parts are multiplied (*exp(i) = exp(i)*exp(f)*) to receive the final exponent value of the input argument. These two arguments: *exp(i)* and *exp(f)* are floating point numbers. The first value is get from the precomputed table, the second one is achieved by casting 32 fixed point CORDIC output value to floating point. This is only one floating point operation in our implementation. The Mitrion-C uses dedicated multipliers to implement this operation (Tab. 2).

The value of $\tanh^{-1}(2^{-i})$ is pre-computed for each i in CORDIC algorithm. These values are used in our implementation to create a look-up table.



Rys. 3. Implementacja algorytmu CORDIC
Fig. 3. Cordic implementation

Very often it is necessary in algorithms to compute a lot of exponents for different arguments. In Mitrion-C such computations can be pipelined using *foreach* loop.

Tab. 2 shows the results of implementation of the CORDIC algorithm in Mitrion-C language. The algorithm was implemented on Virtex 4 LX200 with frequency 50 Mhz. The achieved latency is equal to 110 clock cycles. Our implementation uses one FPGA circuit on SGI Altix 4700 machine. It is possible wtih additional license of Mitrion-C to implement the second circuit and run the CORDIC algorithm simultaneously.

In Tab. 3 there is a comparison with two implementations of double precision exponential functions in FPGA, both in VHDL. The first one [9] is implementation of the CORDIC algorithm and gives results in 258 clock cycles with frequency equal to 100 MHz. The second one is implementation of the combined table and polynomial approximation. This algorithm achieved best results in exponent implementation in FPGA (latency 27 clock cycles with 166 MHz). To achieve the double precision accuracy in our solution, computation of the fractional part must have 62 iterations (32 in floating point) [5]. The multiplication of (*exp(i) = exp(i)*exp(f)*) must be made in the double precision format. The latency of such a solution is described in Tab.3. The integer part is computed in the same time but consumes more resources because of more precomputed values. Therefore the latency is about two times higher than in the single precision format (Tab. 3).

Tab. 2. Wyniki implementacji algorytmu CORDIC w Mitrion-C
Tab. 2. Results of CORDIC algorithm implementation in Mitrion-C (floating point)

FPGA	XC4VLX200
Flip Flop	20%
BlockRAM	4%
MUL18	4%
Latency	110 clk cyc

Tab. 3. Porównanie wybranych implementacji funkcji eksponenty w FPGA
Tab. 3. Comparison with other implementations of exponential function

	Our values (32-bit)	Our values (64-bit)	Sass[9] (64 bit)	Wielgosz[1] (64 bit)
FPGA	Virtex 4 LX200	Virtex 4 LX200	Virtex 4 XC4VFX60	Virtex 4 LX200
Core frequency	50 MHz	50 MHz	100 MHz	166 Mhz
Latency of core	110 clk cyc	230 clk cyc	258 clk cyc	27 clk cyc

It is worth saying that a lot of HLLs like Mitrion-C do not have built in elementary functions. This is a serious disadvantage of these tools because, as it was mentioned before, a lot of HPC applications use them massively. Our research shows that implementation of elementary functions using the CORDIC algorithm in HLL is rather slower than other implementations in VHDL. It should be noted that using such implementation is necessary while implementing scientific algorithms in HLL.

6. Literatura

[1] Jamro E., Wiatr K., Wielgosz M.: FPGA implementation of 64-bit exponential function for HPC. FPL 2007, p.718-721, 2007.
[2] Hekstra G., Deprettere E.: Floating-point CORDIC. Proceedings of the 11th IEEE Symposium on Computer Arithmetic, p. 130-137, Windsor, Canada, 1993.
[3] Chen C., Chen R.L., Sheu M.H.: A fast additive normalization method for exponential computation. Proceedings of the Euromicro Symposium on Digital Systems Design, p. 286, Washington, DC, USA, 2003.
[4] Valls J., Kuhlmann M., Parhi K.K.: Efficient mapping of cordic algorithms on FPGA. IEEE Workshop on Signal Processing Systems, p.336-345, 2000.
[5] Volder J.E.: The cordic trigonometric computing technique. IRE Transactions on Electronic Computers, EC-8, no.3, p.330-334,1959.
[6] Detrey J., Dinechin F.: Parameterized floating point logarithm and exponential functions for FPGA. Microprocessing Microsystems, 31(8), p.537-545, 2007.
[7] SGI Altix 4700 documentation, http://www.sgi.com/products/servers/a;tix/4000
[8] Mohl S.: The Mitrion-c programming language, Mitrionics Inc., 2006, http://www.mitrion.com
[9] Pottathuparambil R., Sass R.. Implementation of a CORDIC bsed Double-Precision Exponential. Proceedings of the Fourth Annual Reconfigurable Systems Summer Institute (RSSI '08), Urbana, Illinois, USA, July 7-9, 2008.
[10] Budyń D., Russek P., Wiatr K.: Performance comparison of hardware languages based on Mitrion-C and VHDL case study for CORDIC algorithm", Pomiary, Automatyka, Kontrola, 2011 vol. 57 nr 8, s.933-935.

otrzymano / received: 02.04.2012
przyjęto do druku / accepted: 01.06.2012 *artykuł recenzowany / revised paper*