Highly efficient structure of 64-bit exponential function implemented in FPGAs

Maciej Wielgosz[1,2], Ernest Jamro[1,2], Kazimierz Wiatr[1,2]

1. AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków
2. Academic Computer Centre Cyfronet AGH,
ul. Nawojki 11, 30-950 Kraków
email: {wielgosz, jamro, wiatr}@agh.edu.pl

Abstract. This paper presents implementation of the double precision exponential function. A novel table-based architecture, together with short Taylor expansion, provides low latency (30 clock cycles) which is comparable to 32-bit implementations. Low area consumption of a single *exp()* module (roughtly 4% of XC4LX200) allows implementation of several parallel modules on a single FPGAs. The exp() function was implemented on the SGI RASC platform, thus external memory interface limitation allowed only a twin module parallelism. Each module is capable of processing at speed of 200 MHz with max. error of 1 ulp, RMSE equals 0,62. This implementation aims primarily to meet quantum chemistry's huge and strict requirements of precision and speed.

Key words: HPRC (High Performance Reconfigurable Computing), FPGA, elementary function, exponent function.

1 Introduction

The HPRC (High Performance Reconfigurable Computing) has important advantages over HPC (High Performance Computing) like significantly lower power consumption and more efficient silicon coverage. Unfortunately conducting floating-point operation within FPGA absorbs much more area than fixed-point calculations, therefore for a long time, FPGAs had not been employed to support double precision operation. Nowadays, there are some implementations of single precision floating-point operations[1,2,3]. Since the proposed *exp()* module aims to speed up HPC chemistry and physics calculations, it has to be compatible with the data format employed so far. Consequently, the IEEE-754 double precision standard is adopted.

2 Architecture of *Exp* module

To evaluate *exp* function the following commonly known mathematical identities are employed:

$$e^x = 2^{x \cdot \log_2 e} = 2^{x_i} \cdot e^{x - x_i / \log_2 e} \quad (1)$$

$$e^{x+y} = e^x \cdot e^y \quad (2)$$

where x_i is an integer part of $x \cdot log_2 e$.
The equation (1) is employed to separate input argument into the integer part x_i and fractional part x_f. Integer part x_i is used directly to evaluate the exponent part of the final result 2^{xi}. Therefore the main problem is evaluation of the fractional part $exp(x_f)$.

The proposed architecture of the *exp* function evaluation employs two methods to evaluate the fractional part:

- Look-Up Table (LUT) based architecture
- polynomial approximation.

The partial results of these two methods are combined employing equation (2). Furthermore, (2) can be also used to divide one large LUT memory into several smaller LUT memories. Therefore employment of (2) is the main idea of the proposed architecture.

The mixed method adoption always leads to the dilemma of the trade-off between LUT memories' size and polynomial part evaluation cost. In the case of *exp* function implementation, increase of LUT memory size results in a decrease of the multiplication area. Nevertheless, analysis of the resources occupied by multipliers and LUT memories has led to the conclusion that employment of Block RAMs (BRAMs) embedded in the FPGAs would be the best solution. Replacement of floating-point multipliers with fixed-point ones further reduces occupied FPGA resources. It is possible because the input data was previously converted into a fixed-point format. Furthermore input data smaller than 2^{-60} may be neglected during the calculation as they have unnoticeable impact on the final result.

In the proposed architecture the polynomial approximation is significantly simplified. According to Taylor-Maclaurin exp(x) can be evaluated as follows:

$$e^x = 1 + x + x^2/2 + x^3/6 + \ldots. \quad (3)$$

In order to disregard $x^2/2$ and higher degree expressions, the input argument must be very small to satisfy maximum mantissa error $< 2^{-54}$ for double precision format. This is satisfied for $x<2^{-27}$. Consequently the most significant 27 bits of input x_f are calculated employing LUT-based methods, the reset less-significant bits are calculated using Taylor-Maclaurin expansion limited to: $e^x \approx 1 + x$. To obtain the final result, the results of the LUT-based algorithm and polynomial approximation are multiplied according to (2). It should be noted that (2) allows to use only LUT-based method, nevertheless employment of the polynomial approximation results in a significant decrease of the number of multipliers and LUTs.

Summing up, the input argument x after conversion to fixed-point format is divided into 5 sections:

1. integer part (11-bit), x_i, which evaluates 2^{Xi} (exponent part of the result),
2. fractional MSB part, x_M, bits $2^{-1} \div 2^{-9}$,
3. fractional middle-bits part, x_D, bits $2^{-10} \div 2^{-18}$,
4. fractional LSB part, x_L, bits $2^{-19} \div 2^{-27}$,
5. fractional Taylor part, x_T, bits 2^{-28}…

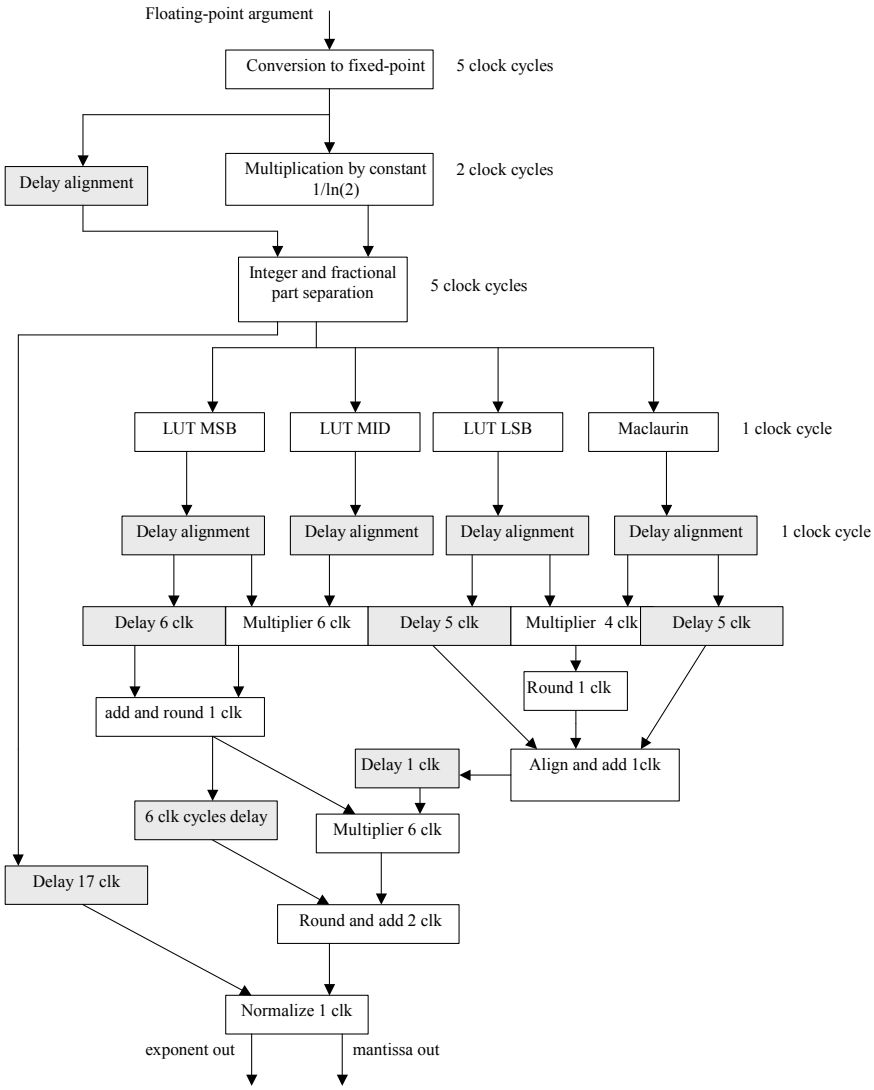

Fig. 1. Exp() module block diagram and its pipeline latency

Afterwards the following mathematical operations are employed:

$$x_I = \lfloor x \cdot \log_2(e) \rfloor \quad (4)$$

$$x_F = x_M \,\&\, x_D \,\&\, x_L \,\&\, x_T = x - x_I \cdot (\log_2(2))^{-1} \quad (5)$$

$$y = 2^{x_I} \cdot e^{x_M} \cdot e^{x_D} \cdot e^{x_L} \cdot (1 + x_T) \quad (6)$$

where: & - bit concatenation, $\lfloor x \rfloor$ - rounding to the greatest integer x_I such that $x_I \leq x$.

Using (4) and (5) enables separation of the integer and fractional parts. This step can be considered to be a scaling process that transforms input data to interval of boundaries at 0 and ln(2). It should be noted that x_I is a small integer represented on 10 bits, otherwise *exp(x)* results in infinity. Therefore this approach in practice replaces a large multiplication (required by identity $e^x = 2^{X/ln(2)}$) with two smaller multiplications, one to compute inaccurately x_I according to (4), second to compute (accurately but with reduced width) $x_I \cdot ln(2)$ according to (5).

3.1 Error analysis

In the proposed architecture the following sources of errors can be distinguished:

1. Taylor series expansion,
2. multiplier (and LUT) width limitation.

The Taylor series expansion is limited only to: $e^x \approx 1 + x$. For $x \to 0$ the expansion error can be approximated by the next omitted expression, i.e. $x^2/2$. As input argument $x_T < 2^{-27}$ the Taylor series expansion error is limited roughly by 2^{-55}. It should be noted that the input value is $x_T \geq 0$, thus the result $y_I = 1 + x_T \geq 1$. Consequently relative error is also $\leq 2^{-55}$. Summing up, Taylor series expansion error is much lower than the double precision format accuracy.

The multiplier inputs are roughly 54-bit wide, therefore the product width is 108-bit wide. Such a bit-width is far beyond the required precision, therefore the LSBs of the product are usually disregarded. As a result, in the proposed architecture, some of the LSBs logic is not implemented at all. Unfortunately, calculation error is much greater for the given architecture. To decrease this error, some additional guard bits are provided. i.e. calculations are carried out on 62-bits. Similarly LUT memory bit-width is extended by additional guard bits, as a single calculation error generated by LUT memory is within required double precision format, nevertheless aggregate whole system error can be outside requirements.

4 Implementation results

The *exp* function was implemented on SGI Reconfigurable Application-Specific Computing (RASC) platform [4]. The presented in Tab. 1 and Tab. 2 implementation results contain *exp* module logic consumption together with RASC core services, essential to provide compatibility with Altix 4700. The RC100 Blade is connected using the low latency NUMALink interconnect to the SGI Altix 4700 Host System, for a rated peak bandwidth of 6.4GB per second.

The RASC RC100 Blade consist of two Virtex-4 LX 200 FPGAs, with 40 MB of SRAM logically organized as two 16MB blocks and an 8MB block. The SRAM are 36-bit QDR devices, thus transferring 128-bit data every clock cycle.

128-bit data vectors are read from one SRAM bank, spread into two substreams consisting of 64-bit each. Every clock cycle (due to pipelining) data is processed by two exponential modules and results are concatenated to 128-bit vector which is

finally written to the second SRAM bank. Afterwards the result is transferred through the NUMAlink to the rest of the system.

Table 1. Implementation results

Implementation results	# 4-input LUT	# flip-flops	# 18-Kb BRAMs
Single exp() module	13,614 (7%)	19,704 (11%)	29 (8%)
Twin exp() module	17,897 (10%)	25,461 (14%)	35 (10%)

Table 2. The RASC system parameters

Max. frequency	200 Mhz
Max. error	1 ulp
Root mean square error	0,6186052
Pipeline latency	30 clk

To compare the calculation speed-up achieved by the RASC, average double precision calculation time per single *exp* function [5] is given in Tab. 3 for Pentium 4 and Itanium processors. It is assumed that processors (Table 3) work at 2 GHz while single FPGA was clocked at 200 MHz. The RASC platform provides two FPGA chips (Xilinx Virtex 4 LX200), that allows to double the calculation rate by employing the second FPGA (this is not taken into account in Fig. 2 and Tab. 3).

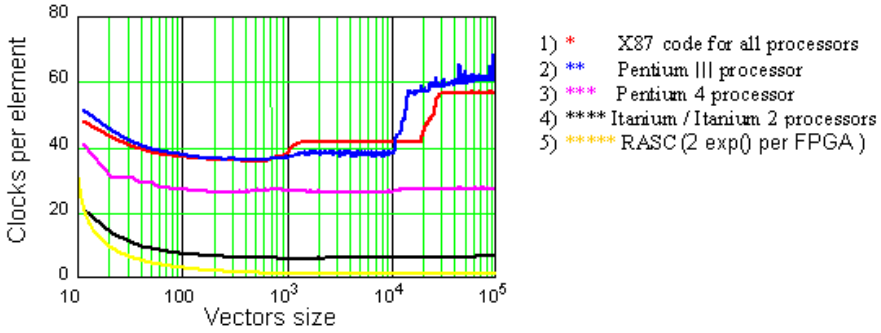

Fig. 2. Exp calculation on different platforms

Table 3. Average calculation time [ns] per an exp calculation

	Pentium 4	Itanium / Itanium 2	RASC
Exp()	13.65	3.08	2.5

The calculation speed-up achieved by the RASC is not significant, nevertheless it should be noted that the throughput can be doubled by employing two FPGAs. Secondly, the calculation throughput is limited by external memory transfers and only 10% of FPGA resources are occupied. Thus additional arithmetic functions can be incorporated in the same FPGA. Besides, by improving external memory interface, the number of parallel *exp* modules can be increased.

It should be noted from Fig. 2 that for general-purpose processors, the calculation time decreases with increasing vector size only up to a curtain limit. Then the calculation time rapidly increases. Probably the reason of this increase is that input or output data cannot be incorporated into internal processor cache memory, and external memory transfers significantly influence the whole system throughput. Summing up, both FPGA and general-purpose processors throughput degradation is caused by external memory access. Nevertheless, this degradation is not taken into account for the general-purpose processors in Tab.3.

6 Summary

This paper describes a novel architecture of double precision exponential function implemented in FPGAs and SGI RASC platform. The presented *exp* architecture introduces several novel hardware solutions never used for *exp* function: a) 3 independent LUTs and Taylor series expansion for *exp* function, b) sign-migration to integer part x_I, fractional part x_F is always positive, c) optimized reduce-width multipliers.

There are two improvements considerations worth introducing. Source code of quantum – chemistry software application can be substantially investigated in the future in order to eliminate a precision overhead. There is still a lot of silicon space on the FPGA (approximately 80%) that can easily fit addition logic. Investigations are being conducted to expand *exp()* function with additional logic of the hot spots found in quantum chemistry application source code.

References

[1] Doss C.C., Riley R.L., Jr., *FPGA-Based Implementation of a Robust IEEE-754 Exponential Unit*, 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), pp. 229- 238

[2] Bui H.T., Tahar S., *Design and Synthesis of an IEEE-754 Exponential Function*, 1999 IEEE Canadian Conference on Electrical and Computer Engineering Shaw Conference Center, Edmonton, Alberta, Canada May 9-12 1999, pp. 450-455 vol.1

[3] Detrey J., de Dinechin F., *A parameterized foating-point exponential function for FPGAs,* IEEE International Conference on Field-Programmable Technology (FPT'05), Singapore, December 2005, pp. 27-34.

[4] Silicon Graphics, Inc., *Reconfigurable Application-Specific Computing User's Guide*, Ver. 004, Mar 2006, SGI

[5] The University of Texas in Austin, *TACC Intel Math Kernel Library,* http://www.tacc.utexas.edu/services/userguides/mkl/functions/exp.html, Nov. 22, 2007