Computing and Informatics, Vol. 35, 2016, 1438–1466

DEDICATED HARDWARE FOR COMPLEX MATHEMATICAL OPERATIONS

Peter Malík

Institute of Informatics
Slovak Academy of Sciences
Dúbravská cesta 9
845 07 Bratislava, Slovakia
e-mail: p.malik@savba.sk

Abstract. New hardware FPGA implementations for the efficient computations of division, natural logarithm and exponential function are proposed. The proposed implementations use generic floating-point adder and multiplier with small additional resources that are shared to compute more frequently used multiply and accumulate operations. Hardware sharing improved the resource utilization. The time of the computation has been reduced to only 6 clock cycles when the natural logarithm and exponential function are calculated. The division is calculated in 5 clock cycles. They are designed as technology independent high throughput computing cores with minimized memory requirements which can be used in higher numbers to significantly increased calculation speed in spectral processing. A new universal arithmetic floating-point unit is also proposed.

Keywords: Dedicated hardware, division, natural logarithm, exponential function, MAC, co-processor, floating-point, FPGA

Mathematics Subject Classification 2010: 65D20, 33F05, 68Q25, 68W25

1 INTRODUCTION

Mathematical operations that require approximation computational techniques have been calculated in hardware since introduction of the first processors. The calculations of square root, natural logarithm, exponential function and other complex operations have been performed by algorithm guided processors interconnected with

a memory. The advancement of semiconductor technology allowed very large scale integrations and systems on chip with many specialized cores implemented on a single chip. The specialized core is optimized to the specific task which is calculated very efficiently with low power and minimized memory requirements. The natural logarithm, exponential function and division are selected for the efficient hardware implementation because they are frequently used in spectroscopy processing.

Many fast algorithms and hardware implementations are studied in literature. A discrete logarithm number system is discussed in [1]. A high-radix composite algorithm for the computation of the logarithm, exponential, and powering functions is used in the proposed architecture in [2]. The logarithm computed in a high-radix digit-recurrence unit with selection by rounding is described in [3]. The same paper discusses the reciprocal obtained by means of a LUT or a digit-recurrence algorithm for fixed-point or floating-point exponents. A fast binary logarithm algorithm is shown in [4]. A Maclaurin series based natural logarithm algorithm is characterized in [5]. Natural logarithm, exponential function and division are discussed in [6]. Fast computing algorithms for the exponential and logarithmic functions are characterized in [7]. A division-free algorithm for fixed-point exponential function based on Newtons method is presented in [8]. Hardware implementations improvements of the Taylor series based exponential function are proposed in [9]. An efficient reducing argument range method for implementation of doubleprecision floating-point is discussed in [10]. The hardware development and implementation of the logarithmic and exponential functions using the CORDIC and parabolic synthesis methodology are studied in [11]. The Newton-Raphson method for fast fixed-point divider is shown in [12] and rounding improvements in [13]. The Goldschmidt division method is discussed in [14] and the version with faster convergence in [15]. A multiply-adder based reduced-precision floating-point unit for embedded digital signal processing with software-based division and square root operations is proposed in [16]. The algorithms differ in convergence rates, computational complexity and memory requirements.

The paper presents new dedicated hardware implementations oriented to the spectroscopy and other signal processing applications. The main motivation is to propose technology independent IP cores capable to compute more complex mathematical operations while the hardware resources are shared for more often used multiply and accumulate (MAC) operations. The proposed IP cores are optimized for the calculation of division, natural logarithm and exponential function. The spectroscopy algorithms frequently use these operations. However, the volume of these operations is small in comparison to MAC. The proposed IP cores use generic floating-point adder and multiplier with small additional resources. The main aim is to use these blocks for calculating the MAC operations. The proposed IP cores are implemented with 32-bit floating-point single precision in FPGA. The dedicated hardware for division is modified to compute division and MACs in the same hardware. The further modifications extended the ability to compute also addition, subtraction and multiplication. The result is 32-bit arithmetic floating-point unit implemented in FPGA. The floating-point representation, originally defined in

IEEE 754-1985 standard and later extended in IEEE 754-2008 standard [17, 18], is widely used in signal processing applications due to the universal use and nearly unlimited interval of input data. The ability to precisely represent values which are very different and far away and the ability to load and process these values with the high precision are crucial in the area of spectroscopy. All proposed hardware implementations are designed with high speed computation and throughput. They are suitable for high computation demanding signal processing applications.

The paper is organized as follows. The mathematical definitions are given in Section 2. Computational models are characterized in Section 3. The dedicated hardware is depicted in Section 4. The specialized processing cores are described in Section 5 and the paper is concluded in Section 6.

2 MATHEMATICAL DEFINITIONS

Many different computational techniques for more complex mathematical operations have been designed. Many of the fast computational techniques are based on the same approximation principle. They differ mostly in the computational complexity and convergence rate.

2.1 Division

The division operation can be implemented as an inverse process of the multiplication operation. The denominator (divisor) is continually subtracted from the numerator (dividend); however, this requires very high number of iterations. If combined with shift operation, the number of iterations is reduced to the number of valid bits of the result. The more precise results require more iterations and therefore fast algorithms were developed, e.g. Newton-Raphson division, Goldschmidt division and its variations. These algorithms converge much faster within one iteration; however, more mathematical operations are needed (usually several additions or subtractions and multiplications).

2.1.1 Newton-Raphson Division

The Newton-Raphson division is based on calculating of multiplicative inverse (reciprocal) of the denominator which is multiplied by the numerator to produce the result [13]. The formula to calculate multiplicative inverse of the denominator is shown in Equation (1) where X_i is multiplicative inverse at iteration i and D is denominator.

$$X_{i+1} = X_i + X_i(1 - DX_i) = X_i(2 - DX_i). \tag{1}$$

The Newton-Raphson division can be optimized by scaling the denominator to the interval $[0.5, 1]$. The minimization of maximal relative error on this interval can be achieved by the initialization shown in Equation (2).

$$X_0 = \frac{48}{17} - \frac{32}{17}D. \tag{2}$$

2.1.2 Goldschmidt Division

The Goldschmidt division is based on the continual multiplication of both the numerator and denominator by the same factor F_i with goal to converge the denominator to 1. At the same time, the numerator converges directly to the quotient [14]. The formula is shown in Equation (3) where Q is quotient, N is numerator, D is denominator and F_i is factor at iteration i.

$$Q = \frac{N}{D}\frac{\prod_i^n F_i}{\prod_i^n F_i}. \tag{3}$$

The factor for the next iteration is calculated by (4). The numerator and denominator for the next iteration are calculated by Equation (5).

$$F_{i+1} = 2 - D_i. \tag{4}$$

$$\frac{N_{i+1}}{D_{i+1}} = \frac{N_i}{D_i}\frac{F_{i+1}}{F_{i+1}}. \tag{5}$$

2.1.3 Goldschmidt Division with Binomial Simplification

The Goldschmidt division can be simplified by the binomial theorem. When scaling N and D such that D is from the interval $(0.5, 1]$, the substitutions (6) and (7) can be used where D is denominator, x is substituted variable and F_i is factor at iteration i.

$$D = 1 - x, \tag{6}$$

$$F_{i+1} = 1 + x^{2^i}. \tag{7}$$

This leads to (8) where N is numerator.

$$\frac{N}{1-x} = \frac{N(1+x)}{1-x^2} = \frac{N\prod_{i=0}^{n-1} 1 + x^{2^i}}{1-x^{2^n}}. \tag{8}$$

2.2 Natural Logarithm

The natural logarithm of positive real number x is the power to which e must be raised to yield x. In other words, the natural logarithm of x is the solution y to:

$$e^y = x. \tag{9}$$

The natural logarithm $\ln(x)$ is the inverse function of the exponential function, leading to the identity [19]:

$$e^{\ln(x)} = x \tag{10}$$

where x is a positive number. The important characteristic of the natural logarithm is mapping multiplication into addition [19]:

$$\ln(xy) = \ln(x) + \ln(y) \tag{11}$$

where x and y are positive numbers. The natural logarithm can be characterized by different techniques. One of the most known definition of the natural logarithm calculation is using the Taylor power series [20] shown in Equation (12):

$$\ln(x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{(x-1)^n}{n}. \tag{12}$$

This definition has a good convergence rate particularly for the small numbers close to 1. More efficient calculation process can be obtained by using power series based on the area hyperbolic tangent function [20]. The principle is shown in Equation (13):

$$\ln(x) = 2 \sum_{n=0}^{\infty} \frac{1}{2n+1} \left(\frac{x-1}{x+1} \right)^{2n+1}. \tag{13}$$

The natural logarithm can be calculated with high precision approximations by arithmetic-geometric mean [21]:

$$\ln(x) \approx \frac{\pi}{2M\left(1, \frac{2^{2-m}}{x}\right)} - m \ln(2) \tag{14}$$

where $M(x, y)$ denotes arithmetic-geometric mean of x and y. The parameter m is chosen such that:

$$x 2^m > 2^{\frac{p}{2}}. \tag{15}$$

where p is the required precision. The continued fractions can be utilized for the calculation of the natural logarithm [22]. The formula is shown in Equation (16):

$$\ln(1+x) = \cfrac{x}{1 + \cfrac{x}{2 - x + \cfrac{2^2 x}{3 - 2x + \cfrac{3^2 x}{4 - 3x + \cfrac{4^2 x}{5 - 4x + \ddots}}}}}. \tag{16}$$

Another formula for the natural logarithm calculation by the generalized continued fractions with the more rapid convergence rate [22] is shown in Equation (17):

$$\ln(1+x) = \cfrac{2x}{2 + x - \cfrac{x^2}{6 + 3x - \cfrac{(2x)^2}{10 + 5x - \cfrac{(3x)^2}{14 + 7x - \ddots}}}}. \tag{17}$$

2.3 Exponential Function

The exponential function e^x can be characterized in different ways. One of the definitions shows that the exponential function e^x is equal to the limit at the infinity [19] defined by (18):

$$e^x = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n \tag{18}$$

where x represents the input variable. The disadvantage of this definition is the slow convergence rate which means the higher precision is calculated with high value of n and the calculation process is slow. Another definition classifies exponential function e^x as solution of y according to the Equation (19):

$$x = \int_1^y \frac{dt}{t}. \tag{19}$$

One of the most known characterization of the exponential function e^x is the definition by power series [23] shown in Equation (20):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}. \tag{20}$$

This definition has good convergence rate particularly for the small numbers close to 0. An yet another exponential function e^x definition can be obtained with Euler identity. The result is the exponential function e^x characterized by continued fraction shown in Equation (21):

$$e^x = 1 + \cfrac{x}{1 - \cfrac{x}{x + 2 - \cfrac{2x}{x + 3 - \cfrac{3x}{x + 4 - \ddots}}}}. \tag{21}$$

The exponential function e^x can be further defined by the generalized continued fraction shown in Equation (22):

$$e^x = 1 + \cfrac{x}{2 - x + \cfrac{x^2}{6 + \cfrac{x^2}{10 + \cfrac{x^2}{14 + \ddots}}}}, \tag{22}$$

which represents the modified continued fraction with the increased convergence rate [22]. It can be shown, from any of these definitions, that the exponential function e^x obeys the basic exponentiation identity defined by (23):

$$e^{x+y} = e^x e^y. \tag{23}$$

3 COMPUTATIONAL MODELS

Computational models are designed to evaluate different computational techniques and to compare the convergence speed. The main question of the study is how many iterations are needed to reach the single and double precision. The convergence is usually dependent on the actual input value. Therefore, it also has to be evaluated. At the end of the evaluation process, the best suited computational technique for the hardware implementation is selected. The main purpose of the computational models is this selection. The computational models are designed in Matlab. Matlab is also used for all evaluation processes.

3.1 Multiplicative Inverse and Division Computational Models

The first mathematical operation to evaluate is the multiplicative inverse. This operation is selected because division can be calculated by multiplication of multiplicative inverse. This is also true in the division computational models which are created from the multiplicative inverse computational models.

3.1.1 Newton-Raphson Multiplicative Inverse and Division Model

The Newton-Raphson multiplicative inverse and division model uses a non-trivial initialization which requires multiplication and subtraction. This results in the initialization process classified as the first iteration cycle. Table 1 shows the relative error of the Newton-Raphson multiplicative inverse model. The convergence rate varies minimally in the interval $[0.5, 1]$ except the boundary values 0.5 and 1. It can be seen that 4 iterations are needed to compute multiplicative inverse with the single precision and 5 iterations are needed to reach the double precision. The division is acquired by additional multiplication with the numerator in the next step. The non zero values in the last column represent limits of the computational models and simulation process which use the double precision.

Input Value	Relative Error at Iteration Cycle				
	1	2	3	4	5
0.50	−0.058824	−0.0034602	−1.1973e−05	−1.4335e−10	0
0.55	−0.016471	−0.00027128	−7.3593e−08	−5.4956e−15	−1.2212e−16
0.60	0.016471	−0.00027128	−7.3593e−08	−5.4623e−15	−1.3323e−16
0.65	0.04	−0.0016	−2.56e−06	−6.5535e−12	1.4433e−16
0.70	0.054118	−0.0029287	−8.5774e−06	−7.3572e−11	0
0.75	0.058824	−0.0034602	−1.1973e−05	−1.4335e−10	1.6653e−16
0.80	0.054118	−0.0029287	−8.5774e−06	−7.3572e−11	0
0.85	0.04	−0.0016	−2.56e−06	−6.5535e−12	−1.8874e−16
0.90	0.016471	−0.00027128	−7.3593e−08	−5.3957e−15	0
0.95	−0.016471	−0.00027128	−7.3593e−08	−5.4845e−15	0
1.00	−0.058824	−0.0034602	−1.1973e−05	−1.4335e−10	0

Table 1. Relative error of the Newton-Raphson multiplicative inverse model

3.1.2 Goldschmidt Multiplicative Inverse and Division Model

The Goldschmidt multiplicative inverse and division model uses trivial initialization. Therefore, it is not classified as an iteration cycle. Table 2 shows the relative error of the Goldschmidt multiplicative inverse model. The convergence rate differs in the interval $[0.5, 1]$. The slowest convergence has the boundary values 0.5, and the opposite boundary value 1 is calculated instantly. This decreasing dependence can be seen in the whole interval $[0.5, 1]$. The computational speed is dependent on the slowest convergence rate and therefore to compute the multiplicative inverse with the single precision 5 iterations are needed and 6 iteration are needed to reach the double precision. The advantage is that the division can be acquired at the same time by changing the initial conditions. The 6^{th} iteration is not shown in Table 2 but the limits of the models and simulation process with the double precision can already be seen in the last column with larger input values.

Input Value	Relative Error at Iteration Cycle				
	1	2	3	4	5
0.50	−0.25	−0.0625	−0.0039063	−1.5259e−05	−2.3283e−10
0.55	−0.2025	−0.041006	−0.0016815	−2.8275e−06	−7.9948e−12
0.60	−0.16	−0.0256	−0.00065536	−4.295e−07	−1.8439e−13
0.65	−0.1225	−0.015006	−0.00022519	−5.0709e−08	−2.5979e−15
0.70	−0.09	−0.0081	−6.561e−05	−4.3047e−09	1.5543e−16
0.75	−0.0625	−0.0039062	−1.5259e−05	−2.3283e−10	0
0.80	−0.04	−0.0016	−2.56e−06	−6.5535e−12	1.7764e−16
0.85	−0.0225	−0.00050625	−2.5629e−07	−6.587e−14	−1.8874e−16
0.90	−0.01	−0.0001	−1e−08	-1.9984e−16	−1.9984e−16
0.95	−0.0025	−6.25e−06	−3.9062e−11	2.1094e−16	2.1094e−16
1.00	0	0	0	0	0

Table 2. Relative error of the Goldschmidt multiplicative inverse model

3.1.3 Multiplicative Inverse and Division Model with Binomial Simplification

The Goldschmidt multiplicative inverse and division model with binomial simplification uses simple initialization with 1 subtraction and therefore it is not classified as an iteration cycle. Table 3 shows the relative error of the binomially simplified Goldschmidt multiplicative inverse model. The convergence rate differs in the interval $[0.5, 1]$ and is very similar to the standard Goldschmidt model. The first three columns are the same, and only the last two columns show small differences in comparison with the standard Goldschmidt model shown in Table 2. Five iterations are needed to compute multiplicative inverse with the single precision and 6 iteration are needed to reach the double precision. The option to perform the division at the same time by changing initial conditions is similarly valid. The standard Goldschmidt model is better due to the trivial initialization. The comparison of the standard Goldschmidt model with the Newton-Raphson model depends on the final application. The standard Goldschmidt model computes the division directly and its non-equal convergence rate can be further utilized. The fastest convergence region near 1 can be targeted by transforming the input data to this region. The transformation can use precomputed values stored in the memory. This makes the standard Goldschmidt model the most suitable computational technique for the hardware implementation.

Input	Relative Error at Iteration Cycle				
Value	1	2	3	4	5
0.50	−0.25	−0.0625	−0.0039063	−1.5259e−05	−2.3283e−10
0.55	−0.2025	−0.041006	−0.0016815	−2.8275e−06	−7.9949e−12
0.60	−0.16	−0.0256	−0.00065536	−4.295e−07	−1.8452e−13
0.65	−0.1225	−0.015006	−0.00022519	−5.0709e−08	−2.5979e−15
0.70	−0.09	−0.0081	−6.561e−05	−4.3047e−09	1.5543e−16
0.75	−0.0625	−0.0039062	−1.5259e−05	−2.3283e−10	0
0.80	−0.04	−0.0016	−2.56e−06	−6.5535e−12	1.7764e−16
0.85	−0.0225	−0.00050625	−2.5629e−07	−6.6058e−14	−3.7748e−16
0.90	−0.01	−0.0001	−1e−08	0	0
0.95	−0.0025	6.25e−06	−3.9063e−11	0	0
1.00	0	0	0	0	0

Table 3. Relative error of the Goldschmidt multiplicative inverse model with binomial simplification

3.2 Natural Logarithm Computational Models

The second mathematical operation to evaluate is the natural logarithm. This operation is selected because it is the most used form of logarithm calculus in signal processing. Five computational techniques defined in the section Natural Logarithm are evaluated.

3.2.1 Taylor Power Series Based Natural Logarithm Model

The Taylor power series natural logarithm definition (12) represents continual addition or subtraction of increasing power values divided by the power index. The relative errors of the Taylor power series model in the specific iteration cycle are shown in Table 4. The first column represents the iteration cycle number and the next five columns show the relative errors of input values $0.5, 0.6, 0.7, 0.8$ and 0.9. The convergence rate of the Taylor power series model is increasing with the decreasing distance to one. Only 7 iterations are needed to calculate the natural logarithm with the single precision of input value 0.9 and 15 iterations to reach the double precision result. However, 22 iterations are needed to calculate the natural logarithm of input value 0.5 to reach the single precision and 47 iterations to reach the double precision. The saturation of the relative error to the non-zero values represents limits of the simulation process with the double precision.

Iteration	Relative Error of Input Value				
	0.5	0.6	0.7	0.8	0.9
1	0.27865	0.21695	0.1589	0.10372	0.050878
2	0.098316	0.060345	0.032733	0.014088	0.0034217
3	0.038203	0.018582	0.0074997	0.0021371	0.00025799
4	0.015661	0.0060535	0.0018222	0.00034455	2.0713e−05
5	0.0066444	0.0020443	0.00045965	5.7742e−05	1.7305e−06
6	0.0028873	0.00070792	0.000119	9.9397e−06	1.4861e−07
7	0.0012772	0.00024973	3.1404e−05	1.7451e−06	1.3023e−08
8	0.00057274	8.9359e−05	8.4105e−06	3.1105e−07	1.159e−09
9	0.00025966	3.2339e−05	2.2789e−06	5.611e−08	1.0441e−10
10	0.00011877	1.1812e−05	6.2337e−07	1.022e−08	9.4997e−12
11	5.4729e−05	4.3474e−06	1.7186e−07	1.8766e−09	8.7157e−13
12	2.5377e−05	1.6104e−06	4.7696e−08	3.469e−10	8.0611e−14
13	1.183e−05	5.9986e−07	1.3312e−08	6.4497e−11	7.5079e−15
14	5.5406e−06	2.2451e−07	3.7336e−09	1.2052e−11	7.903e−16
15	2.6055e−06	8.4378e−08	1.0516e−09	2.2617e−12	1.3172e−16
17	5.8214e−07	1.2046e−08	8.4341e−11	8.0601e−14	1.3172e−16
22	1.4377e−08	9.7187e−11	1.6108e−13	0	1.3172e−16
28	1.7951e−10	3.1731e−13	1.5564e−16	0	1.3172e−16
37	2.6941e−13	0	1.5564e−16	0	1.3172e−16
47	1.6017e−16	0	1.5564e−16	0	1.3172e−16

Table 4. Relative error of the Taylor power series based natural logarithm model

3.2.2 Hyperbolic Tangent Power Series Based Natural Logarithm Model

The hyperbolic tangent power series natural logarithm definition (13) represents continual addition of increasing power values divided by the power index. The relative errors of the hyperbolic tangent power series model in the specific iteration

cycle are shown in Table 5. The first column represents the iteration cycle number and the next five columns show the relative errors of input values $0.5, 0.6, 0.7, 0.8$ and 0.9. The convergence rate of the hyperbolic tangent power series model is increasing with the decreasing distance to one. This increasing tendency is faster in comparison to the Taylor power series model. Only 3 iterations are needed to calculate the natural logarithm with the single precision of input value 0.9 and 6 iterations to reach the double precision result. However, 7 iterations are needed to calculate the natural logarithm of input value 0.5 to reach the single precision and 16 iterations to reach the double precision.

Iter-	Relative Error of Input Value				
ation	0.5	0.6	0.7	0.8	0.9
1	0.038203	0.021192	0.010468	0.0041289	0.00092404
2	0.0025812	0.00080058	0.00019631	3.0628e−05	1.5363e−06
3	0.0002064	3.5887e−05	4.3755e−06	2.703e−07	3.0403e−09
4	1.7923e−05	1.7491e−06	1.0612e−07	2.5968e−09	6.5511e−12
5	1.6348e−06	8.9605e−08	2.7062e−09	2.6239e−11	1.4884e−14
6	1.5408e−07	4.7451e−09	7.1358e−11	2.7427e−13	1.3172e−16
7	1.4865e−08	2.5729e−10	1.9268e−12	2.9852e−15	1.3172e−16
8	1.4596e−09	1.4201e−11	5.2916e−14	1.2438e−16	1.3172e−16
9	1.4528e−10	7.9503e−13	1.4007e−15	1.2438e−16	1.3172e−16
10	1.462e−11	4.5424e−14	0	1.2438e−16	1.3172e−16
11	1.4846e−12	3.0427e−15	0	1.2438e−16	1.3172e−16
12	1.52e−13	6.5202e−16	0	1.2438e−16	1.3172e−16
13	1.5857e−14	4.3468e−16	0	1.2438e−16	1.3172e−16
14	1.9221e−15	4.3468e−16	0	1.2438e−16	1.3172e−16
15	4.8051e−16	4.3468e−16	0	1.2438e−16	1.3172e−16
16	3.2034e−16	4.3468e−16	0	1.2438e−16	1.3172e−16

Table 5. Relative error of the hyperbolic tangent power series based natural logarithm model

3.2.3 Arithmetic-Geometric Mean Based Natural Logarithm Model

The arithmetic-geometric mean natural logarithm approximation (14) represents the simple calculation with utilizing the arithmetic-geometric mean which is obtained by calculating the arithmetic and geometric mean of x and y and let those become x and y in the next iteration. The two numbers converge quickly. The relative errors of the arithmetic-geometric mean model for the single precision in the specific iteration cycle are shown in Table 6. The first column represents the iteration cycle number and the next five columns show the relative errors of input values $0.5, 0.6, 0.7, 0.8$ and 0.9. The results converge more quickly in comparison to power series models and they show no dependence on input values. Six iterations are needed to calculate natural logarithm with the single precision. The double precision converges in 8 iterations; however, the data are not included.

Iter-	Relative Error of Input Value				
ation	0.5	0.6	0.7	0.8	0.9
1	−12.468	−17.275	−25.173	−40.835	−87.602
2	−8.0346	−11.248	−16.528	−27	−58.272
3	−2.1557	−3.1101	−4.6845	−7.814	−17.17
4	−0.083342	−0.12713	−0.2005	−0.34773	−0.79034
5	−0.0001029	−0.0001739	−0.00029838	−0.00055575	−0.0013435
6	3.4665e−09	3.1481e−09	3.0485e−09	3.1806e−09	3.9128e−09
7	3.6222e−09	3.4709e−09	3.7035e−09	4.5867e−09	7.7558e−09
8	3.6222e−09	3.4709e−09	3.7035e−09	4.5867e−09	7.7558e−09

Table 6. Relative error of the arithmetic-geometric mean based natural logarithm model optimized for the single precision

3.2.4 Continued Fractions Based Natural Logarithm Models

The continued fractions and generalized continued fractions natural logarithm definitions (16) and (17), respectively, represent continual denominator division which becomes more and more complex. The relative errors of the continued fractions model are very similar to the Taylor power series model in Table 4. The minimal differences are only at later iterations and therefore the results are not presented. The relative errors of the generalized continued fractions models in the specific iteration cycle are shown in Table 7. The first column represents the iteration cycle number and the next five columns show the relative errors of input values 0.5, 0.6, 0.7, 0.8 and 0.9. The generalized continued fractions model converges faster in comparison to the hyperbolic tangent power series model, see Table 5. Only 3 iterations are needed to calculate the natural logarithm with the single precision of input value 0.9 and 5 iterations to reach the double precision result. However, 5 iterations are needed to calculate the natural logarithm of input value 0.5 to reach the single precision and 11 iterations to reach the double precision.

3.2.5 Evaluation of Natural Logarithm Models

The generalized continued fractions model has the fastest converge rate from all presented models when the input values are close to 1. The disadvantage is the more complex computational process based on divisions. The exactly opposite results are for the Taylor power series model. All results show that the input interval from 0.5 to 1 is too wide for fast natural logarithm computations. The most evaluated computational techniques converge much faster with input value close to 1. This can be achieved by input transformation that utilizes the natural logarithm characteristic (11). This characteristic is also used to calculate the multiplication factor (exponent) and focuses only on the short input interval. The Taylor power series uses only multiplications and additions. A very short input interval close to 1 reduces the polynomial degree necessary to calculate the Taylor

Iter-	Relative Error of Input Value				
ation	0.5	0.6	0.7	0.8	0.9
1	0.038203	0.021192	0.010468	0.0041289	0.00092404
2	0.0012111	0.00036671	8.856e−05	1.3693e−05	6.837e−07
3	3.6771e−05	6.0939e−06	7.2093e−07	4.3751e−08	4.8771e−10
4	1.1009e−06	9.994e−08	5.7942e−09	1.3805e−10	3.4352e−13
5	3.2759e−08	1.6293e−09	4.6299e−11	4.3311e−13	2.6343e−16
6	9.7154e−10	2.6476e−11	3.6886e−13	1.3682e−15	0
7	2.8756e−11	4.2946e−13	2.9571e−15	0	0
8	8.5003e−13	7.1722e−15	0	0	0
9	2.4987e−14	2.1734e−16	0	0	0
10	8.0086e−16	2.1734e−16	0	0	0
11	0	2.1734e−16	0	0	0

Table 7. Relative error of the generalized continued fractions based natural logarithm model

power series with selected precision. It results in the simple and fast computation in comparison to other algorithms (including minimax function or Chebyshev polynomials). Therefore, it is suitable for hardware implementation when it is used only for input values close to 1. The division complexity requires more hardware resources or more time (more iteration steps) to be calculated and therefore the generalized continued fractions model is less suitable for the hardware implementation.

3.3 Exponential Function Computational Models

The third mathematical operation to evaluate is the exponential function. This operation is selected because it is the inverse operation to the natural logarithm and they are often used together in signal processing. The three computational techniques defined in the section Exponential Function are evaluated.

3.3.1 Power Series Based Exponential Function Model

The power series exponential function definition (20) represents the continual addition of increasing power values divided by increasing factorial. The relative errors of the power series model in the specific iteration cycle are shown in Table 8. The first column represents the iteration cycle number and the next five columns show the relative errors of input values $0.1, 0.5, 1, 1.5$ and 2. The convergence rate of the power series model is decreasing with the increasing input absolute value. Only 5 iterations are needed to calculate the exponential function with the single precision of input value 0.1 and 9 iterations to reach the double precision result. However, 13 iterations are needed to calculate the exponential function of input value 2 to reach the single precision and 22 iterations to reach the double precision.

Iter-	Relative Error of Input Value				
ation	0.1	0.5	1.0	1.5	2.0
1	−0.0046788	−0.090204	−0.26424	−0.44217	−0.59399
2	−0.00015465	−0.014388	−0.080301	−0.19115	−0.32332
3	−3.8468e−06	−0.0017516	−0.018988	−0.065642	−0.14288
4	−7.6678e−08	−0.00017212	−0.0036598	−0.018576	−0.052653
5	−1.2749e−09	−1.4165e−05	−0.00059418	−0.004456	−0.016564
6	−1.818e−11	−1.0024e−06	−8.3241e−05	−0.00092599	−0.0045338
7	−2.2723e−13	−6.2197e−08	−1.0249e−05	−0.00016957	−0.0010967
8	−2.8128e−15	−3.4355e−09	−1.1252e−06	−2.7736e−05	−0.00023745
9	−4.0183e−16	−1.7097e−10	−1.1143e−07	−4.0975e−06	−4.6498e−05
10	−4.0183e−16	−7.741e−12	−1.0048e−08	−5.5175e−07	−8.3082e−06
11	−4.0183e−16	−3.2161e−13	−8.3161e−10	−6.8242e−08	−1.3646e−06
12	−4.0183e−16	−1.2525e−14	−6.3598e−11	−7.8033e−09	−2.0735e−07
13	−4.0183e−16	−6.7338e−16	−4.5198e−12	−8.2958e−10	−2.9306e−08
14	−4.0183e−16	−2.6935e−16	−2.9995e−13	−8.2397e−11	−3.8712e−09
15	−4.0183e−16	−2.6935e−16	−1.8624e−14	−7.6787e−12	−4.7997e−10
17	−4.0183e−16	−2.6935e−16	0	−5.5688e−14	−6.1892e−12
19	−4.0183e−16	−2.6935e−16	0	0	−6.4548e−14
22	−4.0183e−16	−2.6935e−16	0	3.9636e−16	−2.404e−16

Table 8. Relative error of the power series based exponential function model

3.3.2 Continued Fraction Based Exponential Function Models

The continued fraction exponential function definition (21) and the generalized continued fraction exponential function definition (22) represent the continual denominator division which becomes more and more complex. The denominator in the generalized continued fraction definition is less complex and uses lower number of additions in comparison to the continued fraction definition. The relative errors of the continued fraction model are very similar to the power series model in Table 8. The minimal differences are only at later iterations and therefore the results are not presented. The relative errors of the generalized continued fraction model in the specific iteration cycle are shown in Table 9. The first column represents the iteration cycle number and the next five columns show the relative errors of input values 0.1, 0.5, 1, 1.5 and 2. The convergence rate is decreasing with the increasing input absolute value much faster in comparison to the power series model. Only 2 iterations are needed to calculate the exponential function with the single precision of input value 0.1 and 4 iterations to reach the double precision result. Six iterations are needed to calculate the exponential function of input value 2 to reach the single precision and 9 iterations to reach the double precision.

Iteration	Relative Error of Input Value				
	0.1	0.5	1.0	1.5	2.0
1	8.3462e−05	0.010884	0.10364	0.56191	∞
2	−1.3897e−08	−4.4048e−05	−0.0014701	−0.011852	−0.052653
3	9.9232e−13	7.8261e−08	1.0312e−05	0.00018484	0.0014811
4	−2.0091e−16	−7.7453e−11	−4.0532e−08	−1.6157e−06	−2.2629e−05
5	0	4.8753e−14	1.0177e−10	9.0645e−09	2.2355e−07
6	0	0	−1.7742e−13	−3.5354e−11	−1.54e−09
7	0	0	0	1.0147e−13	7.8126e−12
8	0	0	−1.6337e−16	−1.9818e−16	−3.0411e−14
9	0	0	−1.6337e−16	1.9818e−16	1.202e−16

Table 9. Relative error of the generalized continued fraction based exponential function model

3.3.3 Evaluation of Exponential Function Models

The generalized continued fractions model has the fastest converge rate from all presented models. The disadvantage is the more complex computational process based on divisions. The power series model uses only multiplications and additions and converges very slowly. All results show that the input interval from −2 to 2 is too wide for fast exponential function computations. All evaluated computational techniques converge much faster with input value close to 0. This can be achieved by input transformation that utilizes the exponentiation identity (23). A very short input interval close to 0 reduces the polynomial degree necessary to calculate the power series with selected precision. It results in the simple and fast computation in comparison to other algorithms (including minimax function or Chebyshev polynomials). Therefore, it is suitable for hardware implementation when it is used only for input values close to 1. The division complexity requires more hardware resources or more time (more iteration steps) to be calculated. This makes the generalized continued fractions model less suitable for hardware implementation.

4 DEDICATED HARDWARE

Every complex algorithm can be divided to the sequence of simple mathematical operations that can be calculated by general processing units. This is an universal solution that is used by many applications. The dedicated hardware offers faster computations, less data traffic and lower power consumption in comparison to general processing units. This is achieved by creating the data flow optimized hardware architecture for the specific task. The operations are grouped together to the more unified computations lengths with better optimized small local memory while parallel computation is introduced. This results in the parallel computation that efficiently utilizes hardware area and requires less access to general memory. The modern design tools from Xilinx or Altera include IP core generators capable to generate specialized IPs which are technology dependent. The higher computation

speed is achieved by using long pipeline, e.g.: Xilinx generated fast divider with the single precision has the latency equaled to 29 clock cycles and natural logarithm with the single precision has the latency equaled to 23 clock cycles. The disadvantage is in no hardware sharing option. The proposed hardware implementations utilize general floating-point multiplier and adder that are aimed to be used for MAC calculations when the computation of division, natural logarithm or exponential function is not needed. This approach introduces hardware resources sharing. MAC operations are much more used in signal processing in comparison to natural logarithm or exponential function. Hardware resources sharing proposes better area utilization leading to higher throughput.

The floating-point hardware implementation requires much more complex computational blocks for multiplication and addition in comparison to the integer hardware implementation. This is caused by maintaining floating-point data format. The advantage is the high precision with general unrestricted input values. Only 32-bit floating-point precision implementation has been selected for the hardware implementation in FPGA. The new reduced 32-bit floating-point adder and multiplier have been designed and implemented. They are technology independent which is the main advantage. The specification of IEEE 754-2008 standard has been simplified by removing support for denormal numbers, similarly as is done in other commercial IP generator design tools. The denormal numbers are allowed as inputs. However, they are internally recognized as zero value. Removing this support has reduced the area of these computational blocks. This results in the increased number of parallel computing blocks and higher throughput. The reduced 32-bit adder and multiplier produce results in one clock cycle. All presenting designs are technology independent. They have been implemented with Xilinx ISE Design suite 14.5 in Xilinx Virtex4 SX35 FPGA with FF668 package. This FPGA contains 4-input look up tables (LUTs) and embedded DPS48 blocks. The utilization of DSP48 blocks significantly reduces the final number of used LUTs and slices in FPGA design. All results are post-place and route results and the verification is done by the post-place and route simulation. VHDL hardware description language is used in all designs.

4.1 Multiplicative Inverse

The Newton-Raphson, Goldschmidt and Goldschmidt with binomial simplification multiplicative inverse models have been designed and implemented. These models require one addition/subtraction and two multiplications to calculate an iteration cycle. Table 10 shows implementation results of these models. The area consumed by computational logic (LUTs) is very similar in all three designs. All three designs have been optimized to high speed computation and throughput. This has been achieved by the parallel computation of several input values at the same time. The Newton-Raphson implementation calculates 3 and the Goldschmidt and Goldschmidt with binomial simplification implementations calculate 2 multiplicative inverses at the same time. This is the reason why the Newton-Raphson implementation has the computational time one and a half times higher than the Goldschmidt

implementation while they both have the same throughput. The Goldschmidt with binomial simplification implementation requires one clock cycle to calculate the initial condition which is calculated by the adder. This results in increased time of the computation to 12 clock cycles and reduced throughput to 0.1667 per clock cycle. The Goldschmidt implementation is the best.

	The Multiplicative Inverse of		
	Newton-Raphson	Goldschmidt	Goldschmidt + Binomial
LUTs as logic	2 352	2 343	2 350
LUTs as route-thru	96	98	103
Total LUTs	2 448	2 441	2 453
Occupied Slices	1 334	1 273	1 282
Flip Flops	291	161	195
Delay	$13 * \text{clk}$	$11 * \text{clk}$	$13 * \text{clk}$
Time of computation	$15 * \text{clk}$	$10 * \text{clk}$	$12 * \text{clk}$
Throughput	0.2/clk	0.2/clk	0.1667/clk
Delay (ns)	14.957	14.990	14.902
Maximum frequency (MHz)	66.858	66.711	67.105

Xilinx Virtex4 SX35 FPGA used

Table 10. Multiplicative inverse implementation in FPGA

The Table 11 shows the relative errors of Newton-Raphson multiplicative inverse 32-bit implementation in FPGA. Input values are shown in the first column. The next 4 columns show the decreasing relative error calculated in each iteration cycle. The decreasing rate is slower in comparison with data shown in Table 1 due to 32-bit physical limitations of used computational blocks. Similarly, the relative errors of Goldschmidt and Goldschmidt with binomial simplification multiplicative inverse 32-bit FPGA implementations are shown in Table 12 and Table 13. The relative errors are calculated from the output values and correct results generated with the double precision in Matlab. The relative error equaled to 1.1011e−7 represents one bit error or 1 ULP (Unit in the Last Place) which corresponds to the mantissa least significant bit negation of the single precision IEEE 754-2008 standard. This value also represents the machine epsilon of the single precision IEEE 754-2008 standard.

Input	Relative Error at Iteration Cycle			
Value	1	2	3	4
0.500	−0.0588	−0.0035	−1.2040e−05	−1.7881e−07
0.625	0.0294	−8.6515e−04	−9.5367e−07	−5.9605e−08
0.750	0.0588	−0.0035	−1.1951e−05	−5.9605e−08
0.825	0.0294	−8.6519e−04	−7.8976e−07	−1.6391e−07

Table 11. Relative errors of the Newton-Raphson multiplicative inverse 32-bit implementation in FPGA

Input	Relative Error at Iteration Cycle				
Value	1	2	3	4	5
0.500	−0.2500	−0.0625	−0.0039	−1.5259e−05	−5.9605e−08
0.625	−0.1406	−0.0198	−3.9107e−04	−2.8312e−07	−5.9605e−08
0.750	−0.0625	−0.0039	−1.5259e−05	−5.9605e−08	−5.9605e−08
0.825	−0.0156	−2.4414e−04	−5.9605e−08	−5.9605e−08	−5.9605e−08

Table 12. Relative errors of the Goldschmidt multiplicative inverse 32-bit implementation in FPGA

Input	Relative Error at Iteration Cycle				
Value	1	2	3	4	5
0.500	−0.2500	−0.0625	−0.0039	−1.5259e−05	−5.9605e−08
0.625	−0.1406	−0.0198	−3.9107e−04	−2.8312e−07	−5.9605e−08
0.750	−0.0625	−0.0039	−1.5259e−05	−5.9605e−08	−5.9605e−08
0.825	−0.0156	−2.4414e−04	−5.9605e−08	−5.9605e−08	−5.9605e−08

Table 13. Relative errors of the Goldschmidt with binomial simplification multiplicative inverse 32-bit implementation in FPGA

4.2 Division

The Goldschmidt division models have been designed and implemented in two 32-bit floating-point versions. The first version represents a minimal modification by introducing the nominator value as general input. The second version, marked as universal Goldschmidt, is additionally modified to correctly calculate the division results from all input values of the single precision IEEE 754-2008 standard including infinity and zero. The division by zero generates infinity result and error signal. All these models require one addition/subtraction and two multiplications to calculate an iteration cycle. Table 14 shows implementation results of these models. The second and third columns correspond to implementation with no DSP48s and the fourth column to implementation with smart DSP48s utilization. The area consumed by computational logic (LUTs) is only slightly bigger in the universal Goldschmidt divider. However, the use of DSP48 blocks significantly reduces the final area. The number of used registers (Flip Flops) is slightly increased which results in the small improvements of delay and computational speed.

4.3 Natural Logarithm

The Taylor power series natural logarithm based model has been designed and implemented in five 32-bit floating-point versions. All of them require one addition/subtraction and one multiplication to calculate an iteration cycle. The first version is designed based on the Taylor power series based natural logarithm model with only limited input value transformation. The computation is rendered with input data from interval from 0.5 to 1 while the input transformation processes the

	Goldschmidt Divider	Universal Goldschmidt Divider	Universal Goldschmidt Divider
LUTs as logic	2 374	2 406	1 274
LUTs as route-thru	98	106	3
Total LUTs	2 472	2 512	1 277
Occupied Slices	1 285	1 314	688
Flip Flops	193	216	217
DSP48s	0	0	8
Delay	$11 * clk$	$11 * clk$	$11 * clk$
Time of computation	$10 * clk$	$10 * clk$	$10 * clk$
Throughput	0.2/clk	0.2/clk	0.2/clk
Delay (ns)	14.892	13.981	13.964
Maximum frequency (MHz)	67.150	71.526	71.693

Xilinx Virtex4 SX35 FPGA used

Table 14. Division implementation in FPGA

other values and special values as infinity. The corresponding output transformation restores the correct results. Therefore, the input value can be any number in 32-bit IEEE 754-2008 format. The implementation results are shown in the second column in Table 15. It can be seen that the limited input value transformation represents slow computation with delay equaled to 50 clock cycles and computational speed equaled to 23 clock cycles. The internal pipeline structure computes two natural logarithm concurrently and therefore the new natural logarithm value is calculated on average in 23 clock cycles when at least 2 input values are in a sequence. The 2^{nd}, 3^{rd} and 4^{th} versions of the Taylor power series natural logarithm based model gradually improve the input value transformation and reduce the core computational interval. The implementation results are shown in the last 3 columns in Table 15. It can be seen that the time of the computation is gradually reduced from 50 to only 9 clock cycles in version 4. The additional optimizations reduced the computational logic area to 2043 LUTs in version 2. However, the more powerful input transformation increased the computational logic area to 2183 LUTs in version 4.

The 5^{th} version of the Taylor power series natural logarithm based model further improves the input value transformation and reduces the core computational interval. The input transformation has been divided to 2 stages to limit the increase of the computational logic area. The more stages have the multiplicative effect. Each stage requires one clock cycle which results in the slower computation of the input value transformation. This increases the overall time of the computation by 2 clock cycles. The implementation results are shown in Table 16. The second column corresponds to implementation with no DSP48s and the third column to implementation with smart DSP48s utilization. The computational logic consumes area corresponding to 2 274 LUTs and the internal registers require 289 Flip Flops. The overall delay is reduced to 14 clock cycles and the time of the computation is

	Taylor Version 1	Taylor Version 2	Taylor Version 3	Taylor Version 4
LUTs as logic	2 124	2 043	2 059	2 183
LUTs as route-thru	67	67	67	67
Total LUTs	2 191	2 110	2 126	2 250
Occupied Slices	1 203	1 116	1 126	1 198
Flip Flops	277	213	216	228
DSP48s	0	0	0	0
Delay	$50 * clk$	$33 * clk$	$25 * clk$	$19 * clk$
Time of computation	$23 * clk$	$16 * clk$	$12 * clk$	$9 * clk$
Throughput	0.043/clk	0.063/clk	0.083/clk	0.111/clk
Delay (ns)	16.928	16.691	17.767	17.828
Maximal frequency (MHz)	59.074	59.913	56.284	56.092

Xilinx Virtex4 SX35 FPGA used

Table 15. Natural logarithm 32-bit floating-point implementations in FPGA

equaled to only 6 clock cycles when at least 2 input data are in a sequence. The input value can be any number in 32-bit IEEE 754-2008 format.

	Taylor Version 5	Taylor Version 5
LUTs as logic	2 274	1 731
LUTs as route-thru	51	2
Total LUTs	2 325	1 733
Occupied Slices	1 264	975
Flip Flops	289	281
DSP48s	0	4
Delay	$14 * clk$	$14 * clk$
Time of computation	$6 * clk$	$6 * clk$
Throughput	0.167/clk	0.167/clk
Delay (ns)	16.568	16.941
Maximal frequency (MHz)	60.357	59.028

Xilinx Virtex4 SX35 FPGA used

Table 16. Natural logarithm 32-bit floating-point implementation in FPGA

The comparison of the Taylor power series based natural logarithm implementations in FPGA are shown in Table 17. The second and the third columns compare the Taylor version 5 with Taylor version 4, and the last two columns compare Taylor version 5 with Taylor version 1. The area consumed by logic is increased by 4.17 % and 7.06 % in comparison to the version 4 and version 1, respectively. The registers are increased by 26.75 % and 4.33 % in comparison to the version 4 and version 1. This area increase is more than balanced with the throughput increase which has risen by 50 % and more than 288 % in comparison to the version 4 and version 1.

Table 18 shows the relative errors comparison of all five 32-bit floating-point FPGA implementation versions of the Taylor power series based natural logarithm.

	Comparison of Version 5 & Version 4		Comparison of Version 5 & Version 1	
LUTs as logic	91	4.17 %	150	7.06 %
LUTs as route-thru	−16	−23.88 %	−16	−23.88 %
Total LUTs	75	3.33 %	134	6.12 %
Occupied Slices	66	5.51 %	61	5.07 %
Flip Flops	61	26.75 %	12	4.33 %
Delay	$-5 * clk$	−26.32 %	$-36 * clk$	−72.00 %
Time of computation	$-3 * clk$	−33.33 %	$-17 * clk$	−73.91 %
Throughput	0.056/clk	50 %	0.124/clk	288.4 %
Delay (ns)	−1.26	−7.08 %	−0.36	−2.13 %
Maximal frequency (MHz)	4.265	7.60 %	1.28	2.17 %

Xilinx Virtex4 SX35 FPGA used

Table 17. Comparison of natural logarithm 32-bit floating-point implementations in FPGA

The input data in decimal formatting are shown in the first column. The 2^{nd}, 3^{rd}, 4^{th}, 5^{th} and 6^{th} columns show the relative errors corresponding to the version 1, version 2, version 3, version 4 and version 5, respectively. The relative errors are calculated from the output values and correct results generated with the double precision in Matlab. It can be seen that the differences in the relative errors are minimal. The highest relative errors are produced with input values very close to 1 which is caused by calculation of the relative error from very small numbers close to 0. The error absolute value can be neglected. The input values greatly distanced from 1 produce nearly no relative error. The relative error equaled to 1.1011e−7 represents 1-bit error or 1 ULP which corresponds to the mantissa least significant bit negation of the single precision IEEE 754-2008 standard.

4.4 Exponential Function

The power series based exponential function model has been designed and implemented in four 32-bit floating-point versions. The first 3 versions require one addition/subtraction and two multiplications to calculate an iteration cycle. The first version is designed based on the power series based exponential function model with no input value transformation. The computation is optimized for input data from the interval from −2 to 2. The input values outside of this interval are computed correctly. However, the computational precision decreases with the absolute input value. The implementation results are shown in the second column in Table 19. It can be seen that no input value transformation represents the slow computation equal to 15 clock cycles and delay equal to 17 clock cycles. The 2^{nd} and 3^{rd} versions of the power series based exponential function model introduce input value transformation which limit the core computational interval. The implementation results are shown in the last 2 columns in Table 19. It can be seen that the time of the computation is gradually reduced from 15 to 11 clock cycles in the version 3 while

Decimal Input Values	Relative Errors of 32-Bit Power Series				
	Version 1	Version 2	Version 3	Version 4	Version 5
0.50	8.3243e−08	−2.7478e−09	−2.7478e−09	−8.8739e−08	−2.7478e−09
0.55	−5.0402e−08	4.9299e−08	4.9299e−08	−5.0402e−08	−3.8576e−07
0.60	2.1333e−07	−2.004e−08	9.6643e−08	9.6643e−08	−3.3119e−07
0.65	4.482e−09	−2.7225e−07	−2.7225e−07	−1.3388e−07	−1.8710e−07
0.70	−1.5437e−07	−1.5437e−07	−1.5437e−07	−1.5437e−07	−2.3068e−08
0.75	5.5703e−08	5.5703e−08	−1.5149e−07	−1.5149e−07	−1.5149e−07
0.80	−1.8447e−07	−1.8447e−07	−1.8447e−07	−2.5125e−07	−9.8581e−07
0.85	2.5096e−07	2.5096e−07	4.3434e−07	4.3434e−07	−8.3852e−07
0.90	−1.3913e−07	−1.3913e−07	−1.3913e−07	−7.7557e−07	−1.3727e−06
0.95	−1.3895e−06	−1.3895e−06	−1.3895e−06	−1.3895e−06	−3.6141e−06
1	5.9605e−08	0	0	−5.9605e−08	0
5	1.8684e−08	−1.2945e−07	−1.2945e−07	−1.2945e−07	−5.5385e−08
125	−8.0075e−08	−8.0075e−08	−8.0075e−08	−8.0075e−08	−8.0075e−08
3e05	−6.1369e−08	−6.1369e−08	−6.1369e−08	−6.1369e−08	−6.1369e−08
5e15	−1.5738e−07	−1.5738e−07	−1.5738e−07	−1.5738e−07	−5.2610e−08
7e30	−3.5913e−08	−3.5913e−08	−3.5913e−08	−3.5913e−08	−3.6429e−08
0.20	−1.8684e−08	−1.8684e−08	−1.8684e−08	−1.8684e−08	−1.7608e−07
3e−02	−1.2581e−08	−1.2581e−08	−1.2581e−08	−1.2581e−08	−1.4219e−07
5e−05	3.5041e−08	3.5041e−08	3.5041e−08	3.5041e−08	−5.8705e−08
7e−15	1.5557e−07	1.5557e−07	1.5557e−07	1.5557e−07	−7.9092e−08
9e−30	4.3288e−08	4.3288e−08	4.3288e−08	4.3288e−08	−7.0835e−08

Table 18. Comparison of five 32-bit floating-point versions of the Taylor power series implementation in FPGA

one clock cycle is used for the input value transformation. The additional optimizations reduced computational logic area to 2559 LUTs in the version 3 which has more powerful input value transformation.

The 4^{th} version of the power series based exponential function model is completely redesigned. One multiplication has been removed. The calculation process requires one addition and one multiplication at each iteration cycle. The computational pipeline is improved and processing of two input values in parallel is introduced. Additionally, the input value transformation is further improved. All input values of 32-bit IEEE 754-2008 format are computed with the full single floating-point precision. This has been achieved by using the 3 stage input value transformation. The transformation requires memory for storing the necessary constants. The memory requirements increase with the quadratic dependency while the effect of the transformation increases only with the linear dependency. This problem have been solved by segmenting the transformation into 3 separate parts that are calculated consequently each one in 1 clock cycle. Each part uses different constants, while the number of all used constants and corresponding computational logic area are minimized. The final results are restored by the output transforma-

	Power Series Version 1	Power Series Version 2	Power Series Version 3
LUTs as logic	2 498	2 585	2 559
LUTs as route-thru	117	94	96
Total LUTs	2 615	2 679	2 655
Occupied Slices	1 350	1 380	1 351
Flip Flops	141	136	141
Delay	17 ∗ clk	15 ∗ clk	13 ∗ clk
Time of computation	15 ∗ clk	15 ∗ clk	11 ∗ clk
Throughput	0.067/clk	0.067/clk	0.091/clk
Delay (ns)	14.957	15.335	15.360
Maximal frequency (MHz)	66.858	65.210	65.104

Xilinx Virtex4 SX35 FPGA used

Table 19. Exponential function 32-bit floating-point implementations in FPGA

tion which also requires 3 clock cycles. The implementation results are shown in Table 20. The second column corresponds to implementation with no DSP48s and the third column to implementation with smart DSP48s utilization. The computational logic consumes area corresponding to 2415 LUTs and the internal registers require 229 Flip Flops. The delay is increased to 17 clock cycles because parallel processing of 2 input values has been introduced. The time of the computation is reduced to only 6 clock cycles when at least 2 input data are in a sequence. The third column shows the single precision exponential unit implemented in Xilinx Virtex 5 SX95T-1 FPGA [24]. From the available data, it can be seen, that this implementation is less computationally efficient in comparison to the presented implementations.

	Power Series Version 4	Power Series DSP Version 4	SP-EAU [24]
LUTs as logic	2 415	1 757	n.a.
LUTs as route-thru	64	15	n.a.
Total LUTs	2 479	1 772	1 669
Occupied Slices	1 277	911	724
Flip Flops	229	229	1792
DSP48s	0	4	6
36k block RAM	0	0	2
Delay	17 ∗ clk	17 ∗ clk	212 ∗ clk
Time of computation	6 ∗ clk	6 ∗ clk	n.a.
Throughput	0.167/clk	0.167/clk	n.a.
Delay (ns)	14.234	13.996	3.149
Maximal frequency (MHz)	70.254	71.449	317.6

Xilinx Virtex4 SX35 FPGA used

Table 20. Exponential function 32-bit floating-point implementations in FPGA

The comparison of the power series based exponential function implementations in FPGA are shown in Table 21. The second and the third columns compare the version 4 with the version 3, and the last two columns compare the DSP version 4 with the DSP version 3. The area consumed by logic is decreased by 5.63 % in the version 4 and by 31.34 % in the DSP version 4 in comparison to the version 3. The registers are increased by 62.41 %. This increase improved the delay that has fallen by 7.33 % in the version 4 and 8.88 % in the DSP version 4. The throughput is increased by 83.52 %.

	Comparison of Version 4 & Version 3		Comparison of DSP Version 4 & Version 3	
LUTs as logic	−144	−5.63 %	−802	−31.34 %
LUTs as route-thru	−32	−33.33 %	−81	−84.38 %
Total LUTs	−176	−6.63 %	−883	−33.26 %
Occupied Slices	−74	−5.48 %	−440	−32.57 %
Flip Flops	88	62.41 %	88	62.41 %
Delay	$4 * \text{clk}$	30.77 %	$4 * \text{clk}$	30.77 %
Time of computation	$-5 * \text{clk}$	−45.45 %	$-5 * \text{clk}$	−45.45 %
Throughput	0.076 / clk	83.52 %	0.076 / clk	83.52 %
Delay (ns)	−1.126	−7.33 %	−1.364	−8.88 %
Maximal frequency (MHz)	5.15	7.91 %	6.345	9.75 %

Xilinx Virtex4 SX35 FPGA used

Table 21. Comparison of exponential function 32-bit floating-point implementations in FPGA

Table 22 shows the relative errors comparison of all four 32-bit floating-point FPGA implementation versions of the power series based exponential function. The input data in decimal formatting are shown in the first column. The 2^{nd}, 3^{rd}, 4^{th} and 5^{th} columns show the relative errors corresponding to the version 1, version 2, version 3 and version 4, respectively. The relative errors are calculated from the output values and correct results generated with the double precision in Matlab. It can be seen that the differences in the relative errors are minimal. The lowest relative errors are generated by the version 4 which produces nearly no error. The relative error equaled to 1.1011e−7 represents 1-bit error or 1 ULP which corresponds to the mantissa least significant bit negation of the single precision IEEE 754-2008 standard.

5 SPECIALIZED PROCESSING CORES

Dedicated hardware computational cores are designed for high performance that is achieved by efficient utilizing the computational logic area and memory. This design technique is used in many processors and systems on chips where many small dedicated hardware blocks are accessed on the demand. Many signal processing applications are so complex that a processor cannot compute results in the specified

Decimal Input Values	Relative Errors of 32-Bit Power Series			
	Version 1	Version 2	Version 3	Version 4
−2.00	−8.9374e−07	−2.3311e−07	−4.5332e−07	0
−1.80	−5.9578e−07	−3.2534e−07	−1.4505e−07	−1.0258e−07
−1.60	−3.6668e−07	−3.6668e−07	−2.1907e−07	−4.7615e−08
−1.35	−6.5499e−07	−4.2507e−07	−4.2507e−07	5.8617e−08
−1.15	−4.656e−07	−1.8324e−07	−1.8324e−07	−1.1296e−07
−0.90	−2.1998e−07	−2.1998e−07	−2.1998e−07	−1.7052e−07
−0.70	−4.6861e−07	−4.6861e−07	−2.2855e−07	−1.2044e−07
−0.45	−3.017e−07	−3.017e−07	−3.017e−07	−3.3182e−08
−0.25	−1.499e−07	−1.499e−07	−1.499e−07	3.1717e−09
−0.05	−1.5793e−07	−1.5793e−07	−1.5793e−07	−1.5719e−07
0	0	0	0	0
0.05	−2.4144e−07	−2.4144e−07	−2.4144e−07	−1.2879e−07
0.25	−1.7478e−07	−1.7478e−07	−1.7478e−07	1.0897e−08
0.45	−3.9113e−07	−3.9113e−07	−3.9113e−07	−1.5118e−07
0.70	−5.9623e−07	−5.9623e−07	−3.5944e−07	−2.2913e−07
0.90	−5.0091e−07	−5.0091e−07	−4.0398e−07	−8.9332e−08
1.15	−4.6457e−07	−3.8908e−07	−3.8908e−07	−1.3876e−07
1.35	−3.9143e−07	−2.0601e−07	−2.0601e−07	−1.6805e−07
1.60	−4.6739e−07	−3.7112e−07	−2.7484e−07	−1.0614e−07
1.80	−5.7376e−07	−4.9494e−07	−3.373e−07	−2.1079e−07
2.00	−4.3727e−07	−3.7274e−07	−2.4368e−07	1.4456e−08

Table 22. Precision comparison of four 32-bit floating-point versions of the power series implementation in FPGA

time. The simple access on the demand technique does not utilize the whole hardware area completely. The dedicated hardware computational core can be modified to additionally compute other computationally simpler tasks with original hardware resulting in the more universal specialized processing core which is used very frequently. This approach introduces hardware resources sharing which proposes better area utilization leading to higher throughput.

5.1 Arithmetic Floating-Point Unit

The arithmetic floating-point unit 1 (AFPU1) is the modified dedicated hardware divider with the Goldschmidt division Algorithm (3). It is designed to calculate division and MAC effectively. AFPU1 uses two multipliers and one subtracter of the original divider for calculating MACs. AFPU1 calculates results with 32-bit floating-point precision of IEEE 754-2008 standard. The operation code of AFPU1 and delay values are shown in Table 23. Logical 0 on input signal *operation* selects MAC and logical 1 selects division. MAC has delay equaled to $(n + 2)$ clock cycles, where n is the number of input values. Two division results

are produced after 11 clock cycles. AFPU1 is optimized for the continuous operation.

Operation Code	Operation Type	Delay Values
0	MAC	$(n+2) \times \text{CLK}$
1	division	$11 \times \text{CLK}$

Table 23. AFPU1 operation code and delay values

The arithmetic floating-point unit 2 (AFPU2) is the modified version of AFPU1 with extended ability to calculate addition, subtraction and multiplication. AFPU2 is designed primarily for the effective calculation of division and MAC. The calculations of the other operations utilize only small portions of AFPU2 area. The operation code of AFPU2 and delay values are shown in Table 24. Logical combination 000 or 001 on input signal *operation* selects multiplication. Logical combination 010 or 011 selects MAC. Logical combination 100 selects addition. Logical combination 101 selects subtraction and logical combination 110 or 111 selects division. Addition, subtraction and multiplication are delayed by 2 clock cycles. Other delays are the same as in AFPU1. AFPU2 is optimized for continuous operation.

Operation Code	Operation Type	Delay Values
00x	multiplication	$2 \times \text{CLK}$
01x	MAC	$(n+2) \times \text{CLK}$
100	addition	$2 \times \text{CLK}$
101	subtraction	$2 \times \text{CLK}$
11x	division	$11 \times \text{CLK}$

Table 24. AFPU2 operation code and delay values

Table 25 shows the implementation results of AFPU1 and AFPU2. The second and third columns correspond to the implementations with no DSP48s and the fourth and fifth columns to the implementations with smart DSP48s utilization. The computational logic area (LUTs) of AFPU1 and AFPU2 are increased by 9.9 % and 14.5 % in comparison to the original Goldschmidt divider, see Table 14. It is caused by the additional logic that creates new data flow between original hardware components necessary for computing the additional operations. The new control and signal logic increases the area minimally. The differences in the number of registers (Flip Flops) are minimal. The AFPU1 and APFU2 maximal operational frequencies are decreased by 9 % and 12.3 % in comparison to the original Goldschmidt divider.

6 CONCLUSIONS

The new hardware implementations for efficient computation of division, natural logarithm and exponential function have been presented. They have been imple-

	AFPU1	AFPU2	AFPU1 DSP	AFPU2 DSP
LUTs as logic	2 644	2 756	1 502	1 507
LUTs as route-thru	97	97	3	2
Total LUTs	2 741	2 853	1 505	1 509
Occupied Slices	1 413	1 474	791	798
Flip Flops	207	208	207	207
DSP48s	0	0	8	8
Delay (ns)	15.363	15.947	15.332	15.587
Maximal frequency (MHz)	65.091	62.708	65.223	64.156

Xilinx Virtex4 SX35 FPGA used

Table 25. Arithmetic Floating-point Unit 1 and 2 implementations in FPGA

mented with the 32-bit floating-point single precision in FPGA. The time of the computation has been reduced to only 6 clock cycles when the natural logarithm and exponential function are calculated. The division is calculated in 5 clock cycles. The division implementation has been modified to compute division and MACs. This resulted in the new arithmetic floating-point unit. The further modifications extended support for addition, subtraction and multiplication calculation. The presented implementations are technology independent. They are designed with the high speed computation and throughput. They are oriented to high computational demanding signal processing applications. They are all designed as independent computing cores with minimized memory requirements which can be used in higher numbers to significantly increased calculation speed in spectral processing. All presented implementation use the general single precision multiplier and adder which can be utilized for MAC calculation. This hardware resource sharing will be further evaluated in the future work. The access to the source codes can be found at the website [25].

Acknowledgment

This work has been supported by Slovak national project VEGA 2/0192/15 and ERDF – ITMS 26240220060.

REFERENCES

[1] Fit-Florea, A.—Lin, L.—Thornton, M. A.—Matula, D. W.: A Discrete Logarithm Number System for Integer Arithmetic Modulo 2^k: Algorithms and Lookup Structures. IEEE Transactions on Computers, Vol. 58, 2009, No. 2, pp. 163–174.

[2] Pineiro, J.-A.—Ercegovac, M. D.—Bruguera, J. D.: Algorithm and Architecture for Logarithm, Exponential, and Powering Computation. IEEE Transactions on Computers, Vol. 53, 2004, No. 9, pp. 1085–1096.

[3] Vazquez, A.—Bruguera, J. D.: Iterative Algorithm and Architecture for Exponential, Logarithm, Powering, and Root Extraction. IEEE Transactions on Computers, Vol. 62, 2013, No. 9, pp. 1721–1731.

[4] Turner, C. S.: A Fast Binary Logarithm Algorithm [DSP Tips & Tricks]. IEEE Signal Processing Magazine, Vol. 27, 2010, No. 5, pp. 124–140.

[5] Aroutchelvame, S. M.—Raahemifar, K.: An Efficient Algorithm and Architecture for Natural Logarithm Using Maclaurin Series. Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2005), 2005, Gammarth, Tunisia, pp. 1–4.

[6] Lyons, R. G.: Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook. 2nd Edition. Wiley-IEEE Press, 2012.

[7] Changfeng, Y.—Yongjuan, M.—Jin, X.: A Quick Algorithms of High Precision on the Exponential and Logarithmic Functions. Proceedings of the 2011 IEEE 2nd International Conference on Computing, Control and Industrial Engineering (CCIE), 2011, Wuhan, China, pp. 159–162.

[8] Chang, Ch. H.—Chen, S. H.—Chen, B. W.—Wang, J. Ch.—Wang, J. F.: A division-Free Algorithm for Fixed-Point Power Exponential Function in Embedded System. Proceedings of the International Conference on Orange Technologies (ICOT), Tainan, Taiwan, 2013, pp. 223–226.

[9] Nilsson, P.—Shaik, A. U. R.—Gangarajaiah, R.—Hertz, E.: Hardware Implementation of the Exponential Function Using Taylor Series. Proceedings of the 32nd Norchip Conference The Nordic Microelectronics event, Tampere, Finland, 2014, pp. 1–4.

[10] Wang, L.—Chen, Y.—Huang, S.: Efficient Argument Range Reduction for Implementation of Double-Precision Floating-Point Exponential Function. Proceedings of the Sixth World Congress on Intelligent Control and Automation (WCICA), Dalian, China, 2006, pp. 6800–6803.

[11] Pouyan, P.—Hertz, E.—Nilsson, P.: A VLSI Implementation of Logarithmic and Exponential Functions Using a Novel Parabolic Synthesis Methodology Compared to the CORDIC Algorithm. Proceedings of the 20th European Conference on Circuit Theory and Design (ECCTD), Linkoping, Sweden, 2011, pp. 709–712.

[12] Rodriguez-Garcia, A.—Pizano-Escalante, L.—Parra-Michel, R.—Longoria-Gandara, O. H.—González, J. C.: Fast Fixed-Point Divider Based on Newton-Raphson Method and Piecewise Polynomial Approximation. Proceedings of International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, 2013, pp. 1–6.

[13] Muller, J.-M.: Avoiding Double Roundings in Scaled Newton-Raphson Division. Proceedings of Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, 2013, pp. 396–399.

[14] Kong, I.—Swartzlander, E. E.: A Rounding Method to Reduce the Required Multiplier Precision for Goldschmidt Division. IEEE Transactions on Computers, Vol. 59, 2010, No. 12, pp. 1703–1708.

[15] Kong, I.—Swartzlander, E. E.: A Goldschmidt Division Method with Faster Than Quadratic Convergence. IEEE Transactions on Very Large Scale Integration (VLSI) System, Vol. 19, 2011, No. 4, pp. 696–700.

[16] Viitanen, T.—Jaaskelainen, P.—Takala, J.: Inexpensive Correctly Rounded Floating-Point Division and Square Root with Input Scaling. Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS), Taipei City, 2013, pp. 159–164.

[17] 754-2008 – IEEE Standard for Floating-Point Arithmetic. August 2008, `http://ieeexplore.ieee.org/servlet/opac?punumber=4610933`.

[18] Hollasch, S.: IEEE Standard 754 Floating Point Numbers. `http://steve.hollasch.net/cgindex/coding/ieeefloat.html`. Last update 2015Dec2.

[19] Maor, E.: The Story of a Number. Princeton University Press, Princeton, NJ, 2009.

[20] Abramowitz, M.—Stegun, I. A.: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. 10th Ed., New York, Dover Publications, 1972.

[21] Sasaki, T.—Kanada, Y.: Practically Fast Multiple-Precision Evaluation of $\log(x)$. Journal of Information Processing, Vol. 5, 1982, No. 4, pp. 247–250.

[22] Lorentzen, L.—Waadeland, H.: Continued Fractions. Atlantis Press, 2008.

[23] Rudin, W.: Real and Complex Analysis. 3rd Ed. McGraw-Hill, 1986.

[24] Alachiotis, N.—Stamatakis, A.: FPGA Optimizations for a Pipelined Floating-Point Exponential Unit. Reconfigurable Computing: Architectures, Tools and Applications, Vol. 6578, 2011, pp. 316–327.

[25] Malík, P.: The Access to the Source Codes of the Presented IP Cores. Available at: `http://www.ui.sav.sk/w/en/dep/ddds/app`.

Peter Malík works as a senior researcher at the Institute of Informatics of the Slovak Academy of Sciences in the Department of Design and Diagnostics of Digital Systems. He received his Master's and Ph.D. degrees in electrical engineering in 2004 and 2010 from the Faculty of Electrical Engineering and Information Technology of the Slovak Technical University in Bratislava. His research interests include digital design, FPGA, signal processing, and hardware reliability.