International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 11, November-2014

Implementation of Log and Exponential Function in FPGA

J. Sujitha
Ece Department
Pbr Vits,Kavali

V. Ramohan Reddy
Asst.Professor,Ece Department
Pbr Vits, Kavali.

Abstract: **The present pattern back toward components intense indication handling has discovered a comparative deficiency of knowledge of components signal processing architectures. Many components effective methods are available, but these are usually not well known due to the popularity of software systems over the past one fourth millennium. Among these techniques is a set of shift-add techniques together known as CORDIC for handling a extensive range of functions such as certain trigonometric, hyperbolic and linear functions. This paper presents architecture of CORDIC, embedded with a sealing that has only little variety of adders and shifters. This paper suggests multiplier-less structure for the execution of exponential and logarithmic functions based on CORDIC technique.**

Keywords: CORDIC, Exponential, Logarithmic, FPGA

INTRODUCTION:

The advantages in the very large scale integration (VLSI) Technology and the advent of
electronic design automation tools have been directing the current research in the areas of digital signal processing, communications, etc in terms of the style of high-speed VLSI architectures for actual time algorithms and systems which have applications in the above mentioned areas. The development rate of VLSI technology was predicted by Gordon Moore and since 1965 technologies have been developed by the industry fitting his predicted curve, which was introduced as the so called Moore's law. These have provided momentum to the designers for transforming algorithm into architecture. Many DSP algorithms, like Image enhancement, Segmentation, Image scaling use elementary functions like logarithmic, trigonometric, exponential, division and multiplication.

Two of the ways of implementing these functions are by using table lookup method and through polynomial expansions. The above mentioned methods require large number of multiplications or divisions and additions or subtractions. Coordinate rotation digital computer (CORDIC), a special purpose computer to compute many non-linear transcolental functions, was proposed by Volder in 1959. Additions to the CORDIC concept depending on work by David. Walther and others solutions to a wider type of features[1 2]. As we are aware of that the multiplications take a large area (especially DSP blocks) and high delay in giving output. In order to overcome this problem, we proposed a new method. This paper attempts to the implementation of Hardware efficient and multiplier less exponential and logarithmic function using CORDIC technique[3].In this paper we aim at reducing hardware by using addition and shifting, improving the speed greater than 175MHz.

LITERATURE OF SURVEY:

The available methods to compute the logarithm of a number using digital circuits can be divided in two main groups. On the one hand, we have the look-up table based algorithms and, on the other, iterative methods. The first approach is faster and straightforward, but only useful for low precision. For implementing it, requires large amount of memory for increasing the accuracy. This is due to the size of the look-up table. We only evaluated iterative algorithms that need small look-up tables. The second group is slower, but suitable for high precision. Taylor's series expansion is among the most popular methods to manually compute logarithms, but it has a slow convergence and requires slow operations like the division. Hence, they are slow when no embedded multipliers are available. Many studies explore hybrid implementations that take advantages from both groups.

Our project required an algorithm that could be implemented on FPGAs from any vendor. It should be platform independent. Our algorithm requires less memory and no multiplier at all to implement exponential and logarithm function.

Powering Function:

The complexity of the powering function, x^y (where x is the base and y the exponent),makes very difficult to implement an efficient and accurate operator in a direct way without any range reduction. However it can be reduced to a combination of other operations and calculated straight forward with the transformation:

$$Z= x^y= e^{yxlnx} \quad (1)$$

A direct implementation of this approach with three sub-operators(a logarithm,a multiplier and a exponential) presents three main problems that have to be effectively handled:

1.The enormous complexity of both exponential and logarithm functions. However, the use of table driven methods in combination with range reduction algorithms makes possible their implementation.

2. The computation with a negative base results in Not a Number even though the powering function is defined for negative bases and integer exponents.
3. Equation (1) can lead to a large error in the result. Although the sub-operators were almost exact the relative error from each sub-operator spreads through the equation generating the final large relative error. Extending the precision of the partial results in an effective way will minimize these relative errors.

To the best of our knowledge there are only two previous works focused on the exponential function [4] [5], and only one for the logarithm function [6] (from the same authors of [5]). The first one [4], employs an algorithm that does not exploit the FPGA characteristics, and consequently presents poor performance. The other two implementations [5, 6] are part of a common work and is designed suiting with FPGA flexibility (using internal tailored fixed arithmetic and exploiting the parallelism features of the FPGA) achieving much better results. They are parameterizable implementations that, additionally to single floating point format, also allow smaller exponent and mantissa bit-widths and are both based on input range reduction and table-driven methods to calculate the function in the reduced range. Our *ex* and ln *x* units, based on these units, include the following innovative features:

Single precision floating point arithmetic extension [5, 6] were designed considering only normalized numbers, not denormalized. Additional logic has been introduced to handle denormalized numbers at the output of *ex* and the input of ln *x*.
1. Redesign of units to deal only with single precision. The feature of bit-width configurability of the base designs has been removed. Thus, the resources needed have been reduced because specific units, just for single precision, have been developed.
2. Simplification of constant multiplications. As suggested in [5], conventional multipliers have been removed where the multiplications involved constant coefficients, improving performance and reducing size.
3. Unsigned arithmetic. In [5, 6] internal fixed arithmetic with sign is used. However, some operations (like the ones involving range reduction and calculation of the exponent for the result in *ex*) are consecutive and related, and the sign of the result can be inferred from the input sign. For such operations signed arithmetic has been replaced by unsigned arithmetic with the corresponding logic reduction.
4. Improved pipelining. The speed is enhanced by systematically introducing pipeline stages to the data path of the exponential and logarithm units and their subunits.

The paper [7] explains about the implementation of power and log function based on a simple modification of power series expansion of Taylor series. In power function implementation, the paper aims at reducing the exponent number to a smaller value. It requires a large amount of block ram and hardware multipliers as well. It becomes platform dependent and the clock frequency may vary from vendor to vendor. The degradation in throughput rate is due to the use of 18 X 18 embedded multipliers in it. The powering unit also requires more number of stages which may be reduced further.

In the proposed method, we are going to reduce delay and improve the throughput rate by avoiding the embedded multipliers and block RAMs. In this paper, we are not completely avoid look up tables, but any value of logarithm or exponential can be calculated, by adjusting the look up table values to the desired number.

PROPOSED METHOD:

The proposed method avoids multiplication and division operations, and is thus appropriate for execution in application on processor chips that absence such guidelines (or where the guidelines are slow) or in components on a automated reasoning system or devoted chip. This method is suitable when shifters are available in abundant. It is an extension to the implementation of sine and cosine explained in CORDIC. The proposed algorithm evaluates the power functions for both positive and negative values. There are some always the same by which it is simple to increase. For example, growing by 2n, where n is a beneficial or a damaging integer, can be carried out by basically moving a variety by n locations. The move will be to the remaining (division) if n is positive, to the right (multiplication) if n is adverse. It is nearly as simple to increase by variety of the form $\pm 2^n \pm 1$. These simply involve an add (or) subtract a shift.

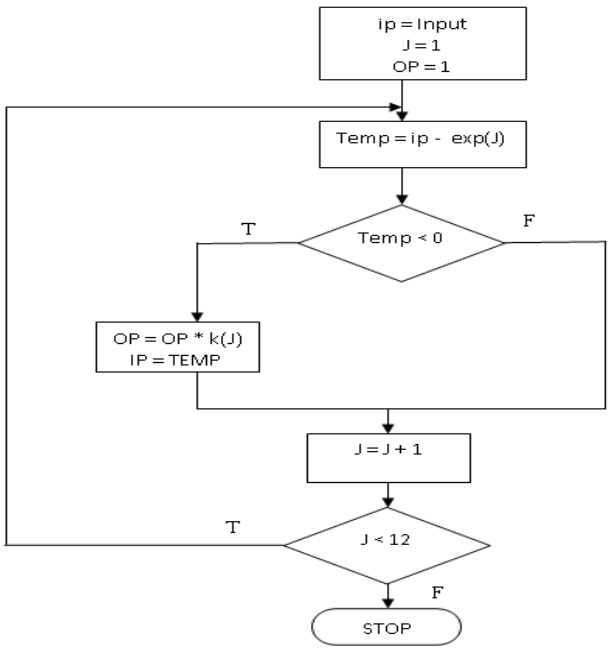
K	Exp(k)
5.5452	256
2.7726	16
1.3863	4
0.6931	2
0.4055	3/2
0.2231	5/4
0.1178	9/8
0.0606	17/16
0.0308	33/32
0.0155	65/64
0.0078	129/128

Implementation of EXP:

For implementing y = exp(x). The criteria is going to produce a series of principles for x and y, and we are going to create sure that for each pair
$y = \exp(4)\cdot$
$y' = \exp(4)\cdot\exp(-(x-k))$
$= \exp(4)\cdot\exp(-x)\cdot\exp(k)$
$= y\cdot\exp(k)$.

In other terms, if we deduct k from x, we have to increase y by exp(k). All we have to do now is create sure that exp(k) is a awesome variety, so we can increase by it quickly, and the relax is uncomplicated. Observe that k itself does not have to be awesome, as we are only subtracting that, not growing by it. Here are some awesome principles of exp(k) and the corresponding (not actually nice) principles of k.

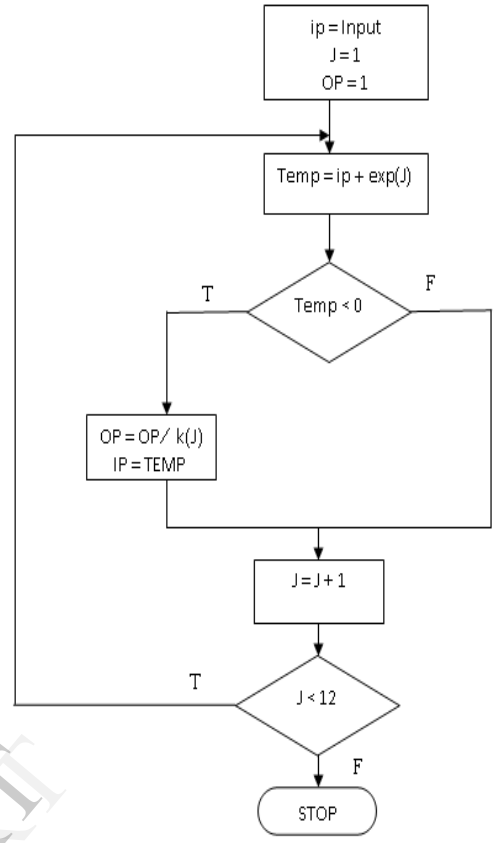
The flow of algorithm is as follows for positive powers of x:



Here in each iteration, we subtract the input from the nearest value of exp(k) as listed in the table. If the difference is negative, we multiply the output by the corresponding exp(k). The process continues withmore entities in our table of k, finally we get the result. In the same way the flow chart is mentioned for nagative powers of x.

K	Exp(k)
5.5452	256
2.7726	16
1.3863	4
0.6931	2
0.2877	3/4
0.1335	7/8
0.0645	15/16
0.0317	31/32
0.0157	63/64
0.0078	127/128
0.0039	255/256

The flow of algorithm is as follows for negative powers of x:



Here in each iteration, we subtract the input from the nearest value of exp(k) as listed in the table. If the difference is positive, we divide the output by the corresponding exp(k). The process continues withmore entities in our table of k, finally we get the result.

Implementation of LOG :

For implementing Y=log (x), As for exp(), the requirements is a sequence of concepts for x and y. Now our invariant is

$\exp(y)\cdot x=54$

or

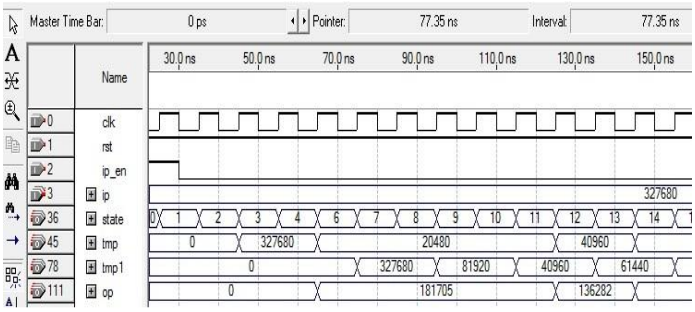
$y=\log(54/x)$.

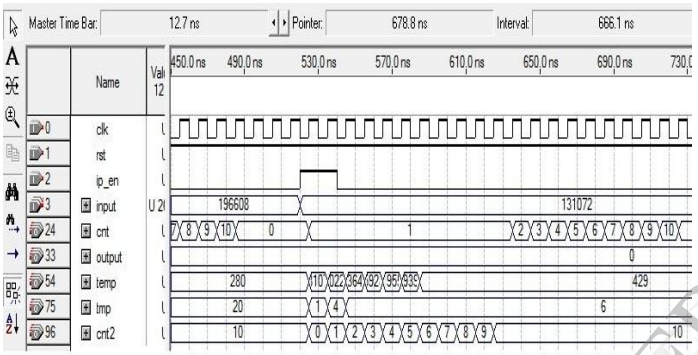
Observe that y=log(54/x)=log(1)=0 as needed. Our aim is to get x to 1 while keeping the invariant. Then y will be given by
y=log(54/1)=log(54),
Suppose we increase x by some variety k. Then to sustain the invariant, the new the invariant, the new y value y′ will have to fulfill
y′=log(54/kx)
=log(54/x)+log(1/k)
=y-log(k).

All the k principles in this desk are higher than 1. We will therefore have to begin with x less than 1 so we begin by growing x by 1/256 (other awesome figures would perform too) after we multiply it by the minimum

value which produces less than 1 and the corresponding output from the lookup table is added to the previous output. This process goes on till the end of the lookup table.

RESULT:





Implementation Issues

LOG

Device	Stratix II
LUT	777
Logic registers	122
Block memory	0
DSP block	0
Clock frequency	207MHz

EXP

LUT	571
Logic registers	53
Block memory	0
DSP block	0
Clock frequency	152MHz

For proving our algorithm, we are using stratix II device and quartus 9.1 edition from ALTERA software. In logarithm implementation, the input is to be multiplied by 65536 and the output we get has to be divided by 65536 in order to get the actual value. The output values of our algorithm are compared with the MATLAB.

While implementing exponential algorithm, we need to multiply input by 65536 to ensure the floating point number converts to a fixed point number. Here we are going to truncate the value to the nearest value. This paper aims at implementing the exp(x) where x value varies from 0 to ±50. But it can be extended further by increasing the bit length required to store the data. But it requires more hardware at the expense of a little delay.

CONCLUSION:

The advantage of the design proposed in this paper is that no DSP or multiplication blocks are used. As 16 bit precision is used, the accuracy of the design is high. The only disadvantage of our approach is that the numbers of iterations required are slightly more. This block can be used in few decoding algorithms in communication systems. The design will be used in LDPC decoder sum product algorithm, image enhancement algorithms the parallel architecture has high throughput (i.e. speed) as compared to serial architecture.

REFERENCES

[1] J. E. Volder, "The CORDIC trigonometric computing technique," IRE Transactions on Electronic Computers, vol. EC- 8, pp. 330–334, Sept. 1959.

[2] B. Gisuthan and T. Srikanthan, "Pipelining flat CORDIC based trigonometric function generators," Microelectronics Journal, volume 33, Pp.77–89, 2002.

[3] E. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC architectures for fast VLSI filtering and array processing," in IEEE International Conference on Acoustic, Speech, Signal Processing, ICASSP'84, March 1984, volume 9, pp.250–253.

[4] C. C. Doss and R. L. Riley, \FPGA-Based execution of a effective IEEE-754 rapid device," in *IEEE Field-Programmable Custom Computing Machines*, 2004, pp.229{238.

[5] J. Detrey and F. de Dinechin, \A parameterized °oating-point exponential function for FPGAs," in *IEEE International Conference Field-Programmable Technology*, 2005, pp.27{34.

[6] ||, \A parameterized °oating-point logarithm operator for FPGAs," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference*,2005, pp. 1186{1190.

[7] Pedro Echeverra, Marisa Lopez-Vallejo,"An FPGA Implementation of the Powering function with Single Precision Floating-Point Arithmetic"