ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2015

Implementation and Comparison of Two Novel Approaches to a Pipelined Logarithmic Multiplier

Shardul P. Telharkar[1], Shantanu P. Telharkar[2], Shreerang G. Dabade[3]

Bachelor or Engineering, Dept. of Electronics and Telecommunications, K J Somaiya Autonomous College of Engineering, University of Mumbai, India[1]

Bachelor of Engineering, Dept. of Electronics, K J Somaiya Autonomous College of Engineering, University of Mumbai, India [2,3]

ABSTRACT: This paper illustrates the implementation of a Logarithmic Multiplier in two novel approaches by modifying a basic architecture. The increasing need of multiplier circuits due to collaboration of the VLSI and signal processing domains requires processors and digital circuits to be fast at the first place. However, many such applications such as video compression allow a minor compromise on accuracy. Logarithmic multipliers provide increased speed, low power consumption and low memory requirement at the expense of accuracy. This paper takes into account two techniques to achieve greater accuracy on a Logarithmic Multiplier without however affecting the speed, power and memory advantages. The final results are compared on the basis of Maximum Error Percentage and Power Dissipation. The proposed architecture as well as the referenced architecture were implemented on Xilinx xc3s1500-5fg676 FPGA and simulated in Verilog on Xilinx 14.5 ISE. Practical implementation was done at 25MHz with signal (toggle) rate of 12.5 %.

KEYWORDS: Mitchell's Algorithm, Pipelining, Signed Multiplier, Error Correction Circuit, Select Logic, Recursive Logic.

I.INTRODUCTION

Multipliers play a major role in various applications like signal processing. Since multiplication is a time consuming, hardware consuming arithmetic function, it proves to be a bottleneck in various mathematical applications. Especially in signal processing applications it gets amplified. Although, in such applications speed is a factor that holds more weightage than accuracy [3], [4], [5], [6]. Moreover, signal processing often deals with distorted signals or improper outputs given by sensors, so inaccurate results are inevitable. Thus, it is preferable to achieve speed at the expense of accuracy. Additional minute errors introduced with multipliers do not affect the results significantly and they can still be acceptable in practice. Thus, methods like logarithmic multiplications are suitable for applications which hold speed of computation more important than the accuracy.

It is imperative to compare such different methods to decide which will yield the best results. This paper sheds light on two such methods of Simple Iterative method known as Signed Multiplier and Recursive method. After implementing the two methods on FPGA, we have based our conclusions on the results achieved. The Recursive method is better in all respects including hardware consumption, speed and power consumption. A tabulated organized conclusion of results obtained will be presented at the end section of this paper
.

II.METHODS USING LOGARITHMIC MULTIPLICATION

In Logarithmic Multiplication the conversion of operand takes places from Integer Number System to Logarithmic Number System (LNS). The multiplication is accomplished in three stages, including the following steps:

1. Calculation of the operand logarithms.
2. Addition of the operand logarithms.
3. Calculation of Antilogarithm of the result.

This method takes advantage of the logarithmic property by the virtue of which multiplication is substituted by addition. LNS multipliers can be generally divided into two categories, one based on methods that use lookup tables and interpolations, and the other based on Mitchell's algorithm (MA) [10]. Generally, Mitchell's Algorithm based methods suppressed lookup tables due to hardware-area savings. However, logarithm and anti-logarithm cannot be calculated exactly. Hence, they need to be approximated, which stands as weakness of this method. The binary representation of the number N can be written as:

$$N=2^k\,(1+\sum_{i=j}^{k-1} 2^{i-k} Z_i) = 2^k(1+k) \qquad (1)$$

Where k is a characteristic number or the place of the most significant bit with the value of '1', Zi is a bit value at the i-th position, x is the fraction or mantissa, and j depends on the number's precision (it is 0 for integer numbers). Owing to the property of logarithm, we choose the base of logarithm as 2. Thus, after taking 2 as the basis of logarithm, the equation reduces to the following,

$$log_2(N) = log_2(2^k(1+x)) = k + log_2(1+x) \qquad (2)$$

Generally, the expression $\log_2(1+x)$ is approximated. Thus, logarithmic-multipliers perform a trade-off between speed and accuracy.

III.MITCHELL'S ALGORITHM BASED MULTIPLIER

Mitchell's Algorithm is one of the most significant methods in Logarithmic Number System [10]. It is employed in applications where the accuracy is not on the highest priority. The binary representation of operands initial equation (1) is used here. But, logarithm and anti-logarithm have to be approximated. The logarithm of the product is:

$$log_2(N_1.N_2) = k_1 + k_2 + log_2(1+x_1) + log_2(1+x_2) \qquad (3)$$

The expression $\log_2(1+x)$ is approximated with x and the logarithm of the two numbers' product is expressed as the sum of their characteristic numbers and mantissas:

$$\log_2(N_1.N_2) \approx k_1 + k_2 + x_1 + x_2 \qquad (4)$$

The characteristic numbers k1 and k2 represent the places of the most significant operands' bits with the value of '1'.
The final MA approximation for the multiplication depends on the carry bit from the sum of the mantissas and is given by:

$$(N_1.N_2)_{MA} = \begin{cases} 2^{k1+k2}(1 + x_1 + x_2), & x_1 + x_2 < 1 \\ 2^{k1+k2+1}(x_1 + x_2), & x_1 + x_2 \geq 1 \end{cases} \qquad (5)$$

The final approximation for the product (5) requires the comparison of the sum of the mantissas with '1'.
The sum of the characteristic numbers determines the most significant bit of the product. The sum of the mantissas is then scaled (shifted left) by 2^{k1+k2} or by $2^{k1+k2+1}$, depending on the x1+x2. If x1+x2 < 1, the sum of mantissas is added to the most significant bit of product to complete the final result. Otherwise, the product is approximated only with the scaled sum of mantissas.

Numerous attempts have been made to improve the MA's accuracy. Hall [5], for example, derived different equations for error correction in the logarithm and antilogarithm approximation in four separate regions, depending on the mantissa value, reducing the average error to 2%, but increasing the complexity of the realization. Abed and Siferd [1], [2] derived correction equations with coefficients that are a power of two, reducing the error and keeping the

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2015

simplicity of the solution. Among the many methods that use look-up tables for error correction in the MA algorithm, McLaren's method [9], which uses a look-up table with 64 correction coefficients calculated in dependence of the mantissas values, can be selected as one that has satisfactory accuracy and complexity. A recent approach for the MA error correction, reducing the number of bits with the value of '1' in mantissas by operand decomposition, was presented by Mahalingam and Rangantathan [8].

IV.SIMPLE ITERATIVE LOGARITHMIC MULTIPLIER

The proposed design presents a better, more simplified way of logarithm approximation introduced in (5). It also introduces an iterative algorithm with several possibilities for achieving miniscule error. In such a methodology, pipelining can be comfortably used boosting the speed of the design and enabling it for high level of parallelism. The correction terms could be calculated almost immediately after the calculation of the product $(N_1.N_2)_{MA}$has been started, once simplification of equation (5) is done.
From equation (1), we can derive a correct expression for the multiplication:

$$P_{true} = N_1.N_2 = 2^{k1}(1 + x_1).2^{k2}(1 + x_2)$$

$$= 2^{(k1+k2)}(1 + x_1 + x_2) + 2^{k1+k2}(x_1x_2) \quad (6)$$

To avoid the approximation error, we have to take into account the next relation derived from (1):
$x.2^k = N - 2^k \quad (7)$

The combination of (6) and (7) gives:
$$P_{true} = N_1.N_2 = 2^{k1+k2} + (N_1 - 2^{k1})2^{k2} + (N_2 - 2^{k2})2^{k1} + (N_1 - 2^{k1}).(N_2 - 2^{k2}) \quad (8)$$

Let
$$P^{(0)}_{approx} = 2^{(k1+k2)} + (N_1 - 2^{k1})2^{k2} + (N_2 - 2^{k2})2^{k1} \quad (9)$$

be the first approximation of the product. It is evident that
$$P_{true} = P^{(0)}_{approx} + (N_1 - 2^{k1}).(N_2 - 2^{k2}) \quad (10)$$

The absolute error after the first approximation is,
$$E^{(0)} = P_{true} - P^{(0)}_{approx} = (N_1 - 2^{k1}).(N_2 - 2^{k2}) \quad (11)$$

We can now add the approximate value of $E^{(0)}$ to the approximate product P_{approx} as a correction term by which we decrease the error of the approximation.
Consider,
$$P_{true} = P^{(0)}_{approx} + C^{(1)} + E^{(1)} \quad (12)$$
$$P^{(1)}_{approx} = P^{(0)}_{approx} + C^{(1)} \quad (13)$$

If we repeat this multiplication procedure with i correction terms, we can approximate the product as
$$P^{(i)}_{approx} = P^{(0)}_{approx} + C^{(1)} + C^{(2)} + \cdots + C^{(i)} = P^{(0)}_{approx} + \sum_{j=1}^{i} C^{(j)} \quad (14)$$

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2015

V.HARDWARE IMPLEMENTATION

We implemented two methods of computing logarithmic multiplication namely, signed multiplication and recursive logic which are mentioned in section B and C of this part.

A. Basic Architecture

As per given in [1], the hardware realization of a 16-bit logarithmic multiplier, consists of a Basic Architecture. This is a 4-stage pipelined architecture that calculates the RHS of the equation (9) re-written below-

$$P^{(0)}_{approx} = 2^{(k1+k2)} + (N_1 - 2^{k1})2^{k2} + (N_2 - 2^{k2})2^{k1}$$

The Basic Architecture consists of two Power evaluation blocks (PEB) (PEB is illustrated in Fig. 1 and 4), two 32-bit barrel shifters, two encoders, two 32-bit adders and a decoder unit. When 2 16-bit numbers are input to this unit, they are passed through a Power Evaluation Block (PEB). This block is used for detecting the leading ones in the input numbers. It gives the power index number k1 which falls between values 0 to 15. Further, a priority encoder outputs the value of k1 and k2, which are then added by a 5-bit adder.

Meanwhile, in the first and the second stage, the value of $(N_1 - 2^{k1})2^{k2}$ and $(N_2 - 2^{k2})2^{k1}$ is calculated using a 32-bit shift left register.

A decoder calculates 2^{k1+k2} which is then input to an adder in the fourth stage that calculates the $P^{(0)}_{approx}$.

B. Signed Multiplier Architecture

The signed numbers' approach only operates with the 2's complement of input numbers. Additional logic required for Fig. 1 Pipelining approach to calculate addition of 3 terms in the equation of $P^{(0)}_{approx}$

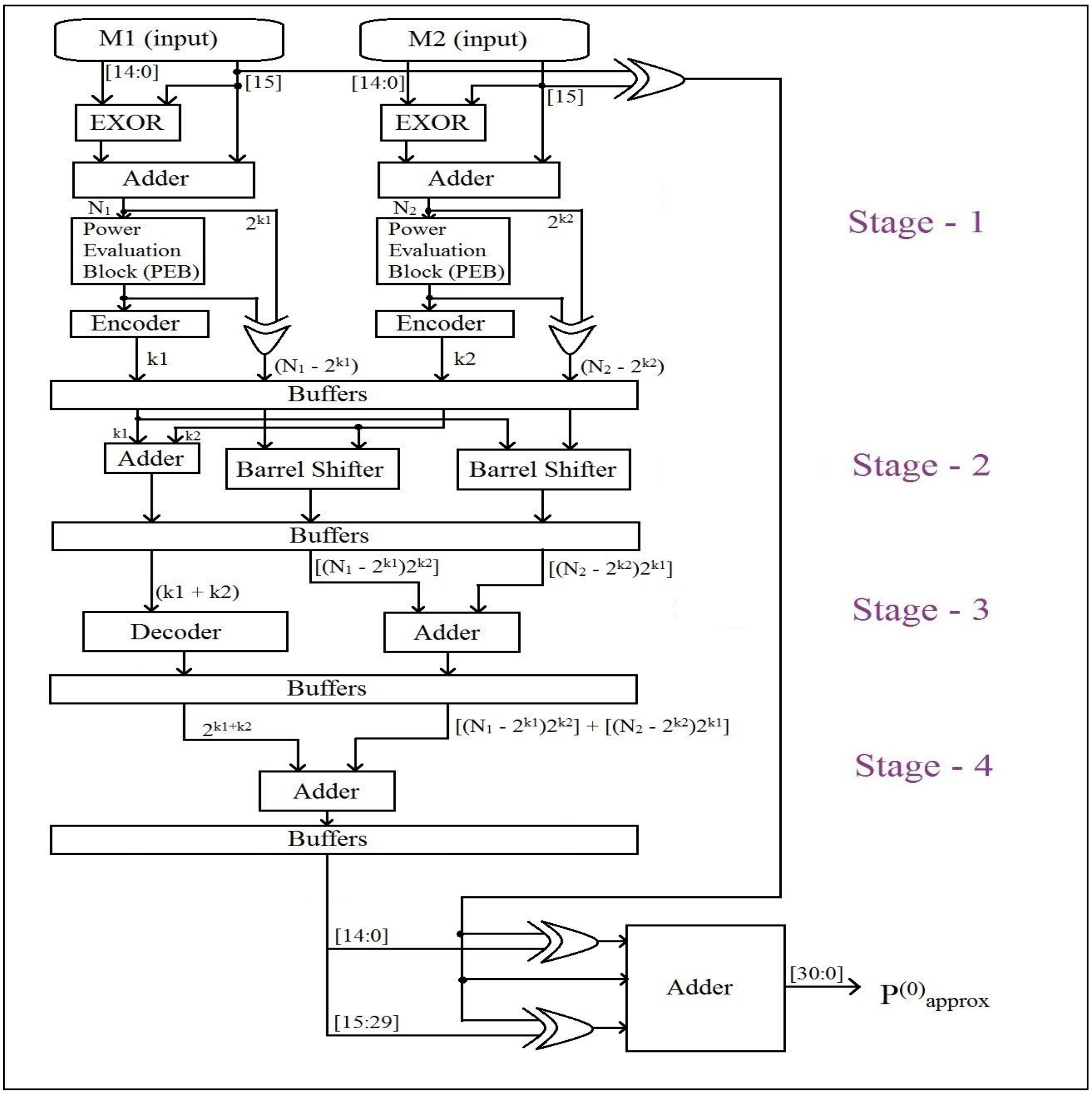
ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2015

the Basic Architecture is given in Fig. 1. It uses a bank of EXOR gates to convert the numbers into unsigned numbers.



The sign bit of final product further depends on the sign bit of both the input numbers. The other architectural features as the placement of buffers for pipelining and the number of pipelining stages is similar to the one referenced earlier. Surprisingly, the power dissipated as well as the relative error both parameters get improved in this method. As shown in Fig 1 below, the number of stages of pipelining remains same in spite of adding the logic. One may add another stage 5 after the final adder and EXOR gates to investigate the performance of the architecture.he proposed architecture has been given below-

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2015

In the implementation of blocks that calculate further approximations, the architecture computes similar to the referenced paper [1]. However, we have put additional circuit at the output end to convert back the 2'complement number into the desired output. The Basic Architecture blocks in Fig 2 below are also known as Error Correction Circuits since they provide the approximations of the error as shown in equation (11). An EXOR bank does this purpose efficiently. We computed approximations up to $P^{(2)}_{approx}$. The stage-wise diagram for the same has been illustrated below.

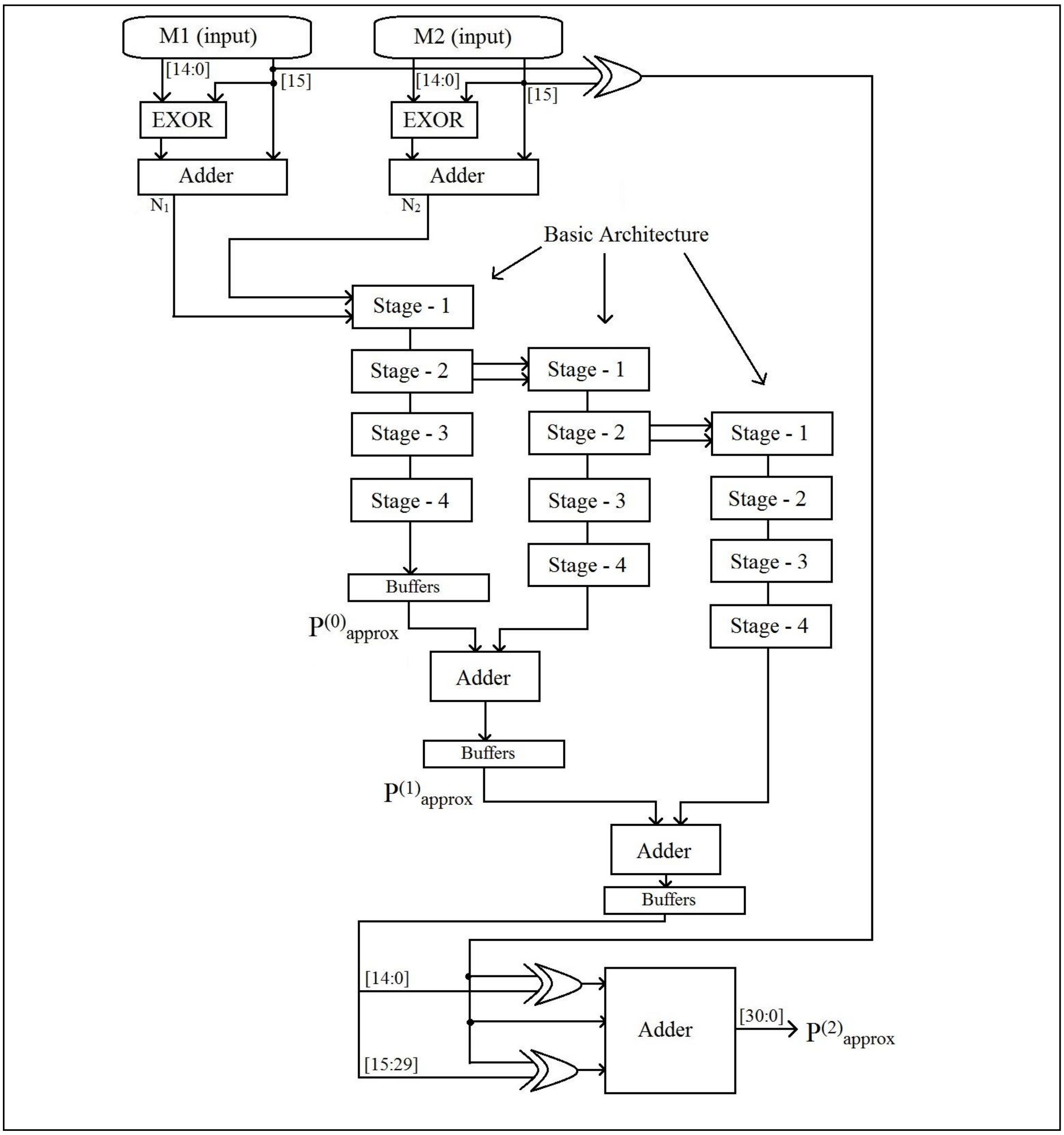


Fig. 2 Complete Iterative Architecture for calculation of 3 successive approximations

C. A novel approach using recursive logic

Basic Idea:
The whole idea behind this approach is making the term $P^{(0)}_{approx}$. equal to P_{true}. For this to happen we must expand and calculate the RHS of the following equation discussed earlier.

$$P^{(i)}_{approx} = P^{(0)}_{approx} + C^{(1)} + C^{(2)} + \cdots + C^{(i)} = P^{(0)}_{approx} + \sum_{j=1}^{i} C^{(j)}$$

To obtain an accurate result, we need to perform those many number of iterations, after which the residue becomes zero. That is $P^{(0)} = P_{true}$. Precisely the number of iterations required is equal to the number of bits with value '1' of that operand which has less number of 1's among the two.
Recursive logic harnesses this fact that by re-performing the iterations till residue is zero we can significantly reduce the average as well as the total error thereby acquiring an accurate output.

Select Logic:
For repeating the calculations, we must have two things. One, a clarity of which input to feedback to the circuit. Second, a select logic that knows when to stop calculating. We have implemented a Select Logic that outputs a status signal, one 16-bit output and requires a select line and a 16-bit input. This block is essentially used to poll continuously whether one of the operand is zero or not. Till that, iterations are necessary and $P^{(0)}$ is not equal to P_{true}. After the residues become zero, the select logic automatically stops the iterations and the circuit is now ready for taking new set of 16-bit input and loads the perfect output into the stage – 5 buffers as shown in the architecture diagram of the multiplier.
The architectural diagram of the Select logic is as follows-

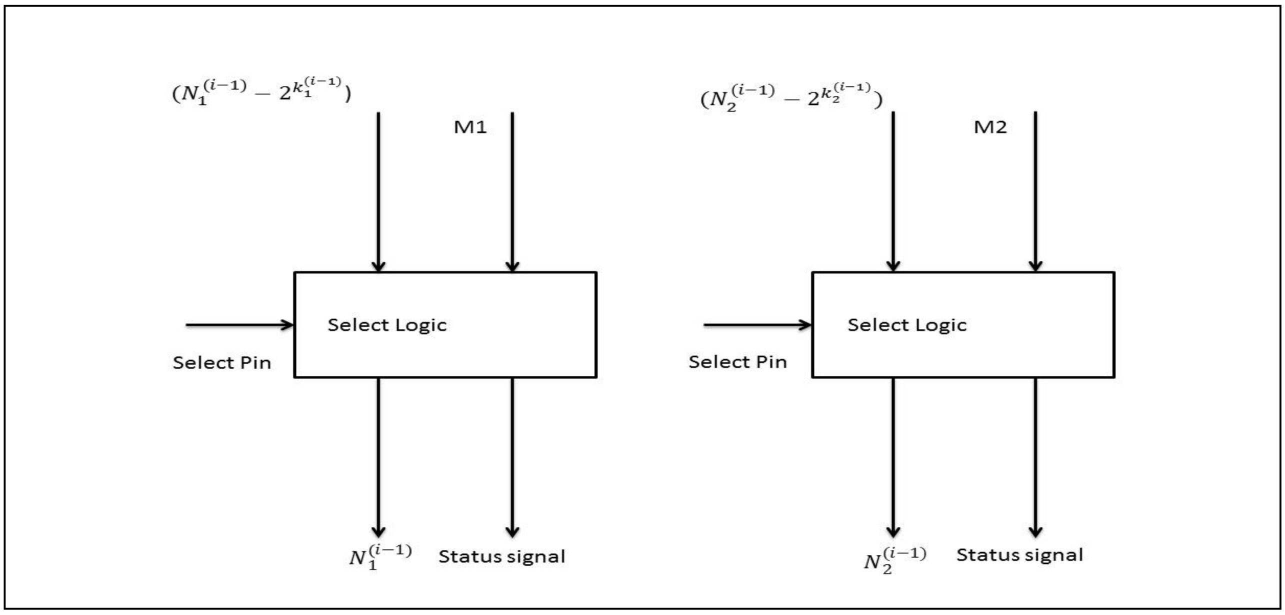


Fig 3 Select Logic Block

Consider 2 inputs M_1 and M_2. At initial stage, the select pin is high, indicating that $N_1 = M_1$ and $N_2 = M_2$. Further, as iterations start, the value of N_1 and N_2 is set equal to the residues.After the 'i' required number of iterations, the value of $(N_1^{(i)} - 2^{k1})$ and$(N_2^{(i)} - 2^{k2})$ approaches zero. At this stage, maximum accuracy is achieved and the circuit no longer needs to carry out iterations. Hence the Select logic now is ready to accept new inputs.

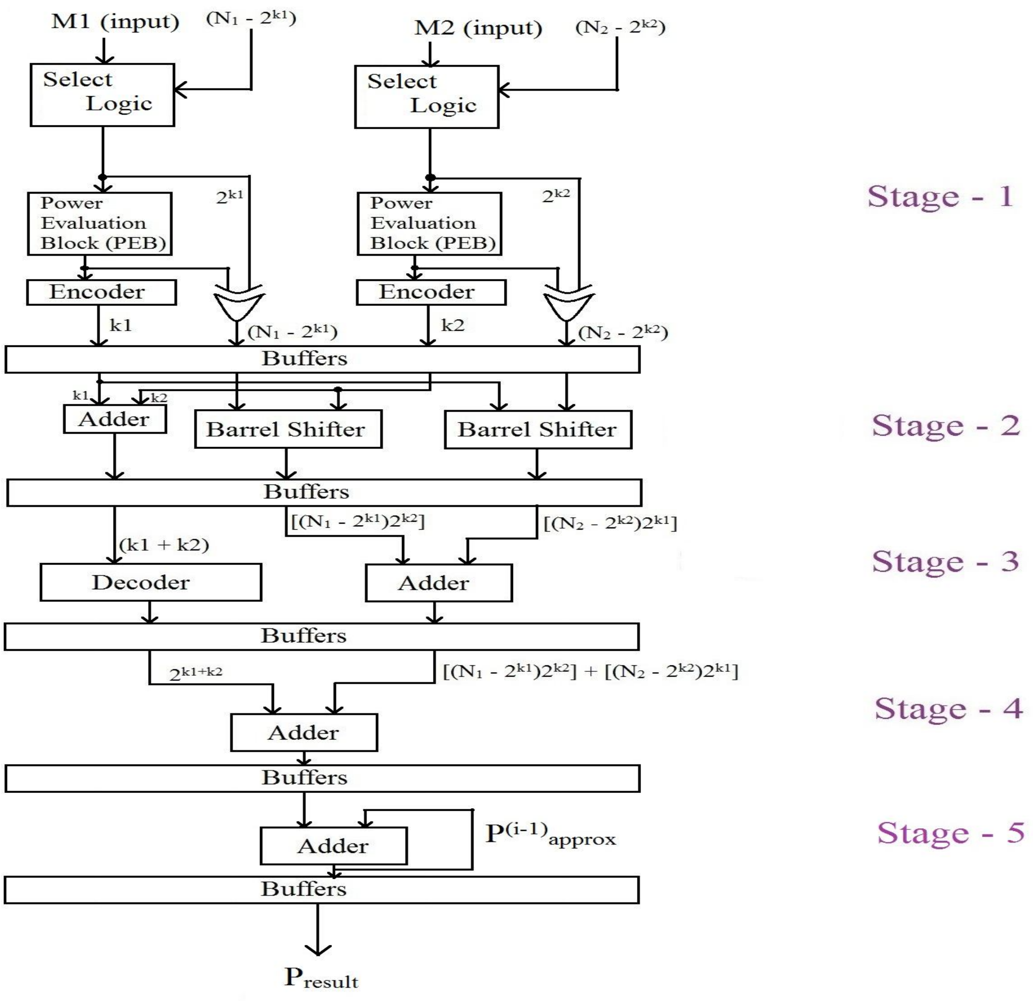
Recursive Approach:
Our design is structured to inculcate recursive approach that feeds back the residues to continue computing approximations. On the final side as well, the previous approximation $P^{i-1}$is fed back to keep adding the approximations.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2015



This additional logic is illustrated in the following diagram. Fundamental blocks such as the four stages and the elementary parts as encoder, decoder, 32-bit barrel shifter and 32-bit adder all remain the same.Fig. 4 Basic Architecture Enhanced with the Recursive Logic Blocks

One unique mention about this block is that it doesn't need to be implemented in the cascaded logic as illustrated in Fig 2. It uses the same hardware for successive iterations. The 3 approximation result of signed multiplier and final result of this recursive logic based multiplier were compared for conclusions.

VI.CONCLUSION AND COMPARISON

In this paper, we have compared two approaches to implement a logarithmic multiplier. After realizing the circuits on Xilinx xc3s1500-5fg676 FPGA, we opine that the recursive approach is more accurate and less power consuming. Since the Error correction circuits are calculated using single block with recursive logic area and power dissipation are greatly reduced. However, even signed multiplier approach is also an efficient when compared with only the basic method discussed in [1].

Total delay required to calculate output depends on number of approximations required before residue becomes zero. This leads to variable output delay. Thus, we conclude that recursive logic is more efficient in terms of accuracy, power consumption and area utilization but is slightly slower as compared to traditional approaches.
Following table provides empirical values of error and power dissipated obtained after successful implementation.

TABLE I

COMPARISON TABLE BASED ON MAXIMUM ERROR IN PERCENTAGE (%)

	E_{rmax}
BA	2.5
BA + 1 ECC	6.25
BA + 2 ECC	1.56
BA + 3 ECC	0.39
Signed (3ECC)	0.18
Recursive Logic	0.00

It can be seen that maximum error percentage decreases to almost half in case of signed and touches zero in case of recursive logic.

TABLE II
COMPARISON BASED ON POWER DISSIPATION AS ACQUIRED FROM XILINX XPOWER

	Logic and Signals (mW)	IO Blocks (mW)	Quiescent (mW)	Total (mW)
BA	3.72	51.26	152.06	207.04
BA + 1 ECC	7.32	51.62	152.66	211.6
BA + 2 ECC	10.66	51.67	153.53	215.36
BA + 3 ECC	13.37	51.93	153.42	218.72
Signed (3ECC)	6.34	49.1	159	214.44
Recursive Logic	4.45	35.67	150.53	190.65

There is an improvement in terms of power dissipation as well. While the total power improvement in signed approach is barely 1.8%, the architecture using recursive logic exhibits an improvement of 12%!

REFERENCES

[1] A Simple Pipelined Logarithmic Multiplier,PatricioBuli´c∗, ZdenkaBabi´c † and AleksejAvramovi´c , Computer Design (ICCD), 2010 IEEE International Conference.

[2] K.H. Abed, R.E. Sifred, VLSI Implementation of a Low-PowerAntilogarithmic Converter, IEEE Transactions on Computers, Vol. 52, No. 9, pp. 1221-1228, September 2003.S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, "A novel ultrathin elevated channel low-temperature poly-Si TFT," *IEEE Electron Device Lett.*, vol. 20, pp. 569–571, Nov. 1999.

[3] L.V. Agostini, I.S. Silva, S. Bampi, Multiplierless and fully pipelinedJPEG compression soft IP targeting FPGAs, Microprocessors and Microsystems, Vol. 31, No. 8, pp. 487-497, December 2007.

[4] V. Gierenz, C. Panis, J. Nurmi, Parameterized MAC unit generation fora scalable embedded DSP core, Microprocessors and Microsystems, Vol. 34, No. 5, pp. 138-150, 2010.

[5] E.L. Hall, D.D. Lynch, S. J. Dwyer III. Generation of Products andQuotients Using Approximate Binary Logarithms for Digital Filtering Applications, IEEE Transactions on Computers, Vol. C-19, No. 2, pp.97-105. February 1970.

[6] J.A. Kalomiros, J. Lygouras, Design and evaluation of a hardware/software FPGA-based system for fast image processing, Microprocessors and Microsystems, Vol. 32, No. 2, pp. 95-106, March 2008.

[7] V. Hampel, P. Sobe, E. Maehle, Experiences with a FPGA-basedReed/Solomon-encoding coprocessor, Microprocessors and Microsystems, Vol. 32, No. 5-6, pp. 313-320, August 2008. "PDCA12-70 data sheet," Opto Speed SA, Mezzovico, Switzerland.

[8] V. Mahalingam, N. Rangantathan, Improving Accuracy in Mitchell'sLogarithmic Multiplication Using Operand Decomposition, IEEE Transactions on Computers, Vol. 55, No. 2, pp. 1523-1535, December2006.

[9] D.J. Mclaren, Improved Mitchell-based logarithmic multiplier forlow-power DSP applications, Proceedings of IEEE International SOCConference 2003 pp. 53-56, 17-20 September 2003.

[10] J.N. Mitchell, Computer multiplication and division using binarylogarithms, IRE Transactions on Electronic Computers, vol. EC-11,pp. 512-517, August 1962.

[11] M.H. Rais. Efficient Hardware Realization of Truncated Multipliersusing FPGA, International Journal of Applied Science, Vol. 5, No. 2,pp. 124 - 128, 2009.

[12] H. Hinkelmann, P. Zipf, J. Li, G. Liu, M. Glesner, On the design of reconfigurable multipliers for integer and Galois fieldmultiplication, Microprocessors and Microsystems 33 (1) (2009) 2–12.