IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 9, Issue 1 Ver. V (Feb. 2014), PP 34-45
www.iosrjournals.org

Design and implementation of modified iterative logarithmic multiplier for low-power and area-efficient applications

Rohan Appasaheb Borgalli[1], Hari Pratap Gautam[2], Winner George Parayil[3]
[1](Information Technology Department, St. Francis Institute of Technology, India)
[2](Electronics and Telecommunications Dept., K.N.G.D Modi Engineering College, India)
[3](Electronics and Telecommunications Dept., St. John College of Engineering and Technology, India)

Abstract: *A multiplier is one of the key hardware blocks in most digital and high performance systems such as FIR filters, digital signal processors and microprocessors etc. many researchers have tried and are trying to design multipliers which offer either of the following- high speed, low power consumption, less area, more accuracy or even combination of them in multiplier.*

This paper presents a simple and efficient logarithmic multiplier with the possibility to achieve a maximum accuracy with less area and low power Consumption through an iterative procedure with recursive logic.

The proposed modified iterative logarithmic multiplier is based on the same form of number representation as Mitchell's algorithm [1962], but for error correction it uses different algorithm proposed by Z. Babic, A. Avramovic , P. Bulic [2011]. And to make it more efficient instead of array of basic block proposed by Z. Babic, A. Avramovic , P. Bulic [2011] it contain only single basic block with recursive logic which finds approximate product and also error correction terms. Due to that it is less area and power consuming in expense of slightly increase in delay. Because there is always trade off between Area and delay.

In order to evaluate the performance of the proposed multiplier and compare it with previous works, we implemented four 16-bit multipliers proposed by Z. Babic, A. Avramovic , P. Bulic [2011]: a pipelined multiplier with no correction terms and three pipelined multipliers with one, two and three correction terms and one 16-bit proposed Modified iterative logarithmic multiplier on the Xilinx xc3s1500-5fg676 FPGA.

Keywords:*Field programmable gate array, Iterative, Logarithmic number system, Multiplier, Mitchell's algorithm,*

I. INTRODUCTION

Multiplier is an electronic circuit used in digital electronics as a key component. A system's performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest clement in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. As a result, many approaches are made towards designing multiplier which provides efficient result. But, Mitchell's algorithm [1] is revolutionary in multiplier design because it utilizes binary logarithms in the operations of multiplication and division. The logarithms used in the arithmetic are approximations to the actual logarithms; because of the approximations, there will be errors in the results of operations using them. It is considered that the simplicity of the method of finding and using these logarithms may make the scheme valuable in some applications. Based on Michelle's algorithm many improvements [2-7] are given to enhance the accuracy of multiplier.

The iterative logarithmic multiplier was proposed by Z. Babic, A. Avramovic, P. Bulic [2].It simplifies the logarithm approximation introduced in [1] and introduces an iterative algorithm with various possibilities for achieving an error as small as required and the possibility of achieving an exact result. This work is further carried out in this paper and minor modification is done for low-power and area-efficient applications.

II. LOGARITHMIC MULTIPLICATION METHODS BASED MULTIPLIER

Logarithmic multiplication introduces an operand conversion from integer number system into the logarithm number system (LNS). The multiplication of the two operands N_1 and N_2 is performed in three phases, calculating the operand logarithms, the addition of the operand logarithms and the calculation of the anti-logarithm, which is equal to the multiple of the two original operands [2] as shown in block diagram Fig-1. The main advantage of this method is the substitution of the multiplication with addition, after the conversion of the operands into logarithms. 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) [1], although there is a lookup-table approach in some of the MA-based methods [3]. Generally, MA-based methods suppressed lookup tables due to hardware-area savings. However, this simple idea has a significant weakness:

logarithm and anti-logarithm cannot be calculated exactly, so there is a need to approximate the logarithm and the antilogarithm [2]. The binary representation of the number *N* can be written as:

$$N = 2^K(1+\sum_{i=j}^{K-1} 2^{i-K}Z_i) = 2^K(1+X) \qquad (1.1)$$

where, *K* is a characteristic number or the place of the most significant bit with the value of '1', Z_i 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). The logarithm with the basis 2 of *N* is then:

$$\begin{aligned}\log_2(N) &= \log_2(2^K(1+\sum_{i=j}^{K-1} 2^{i-K}Z_i)) \\ &= \log_2(2^K(1+X)) \\ &= K+\log_2(1+X)\end{aligned} \qquad (1.2)$$

The expression $log_2(1+x)$ is usually approximated; therefore, logarithmic based solutions are a trade-off between the time consumption and the accuracy.

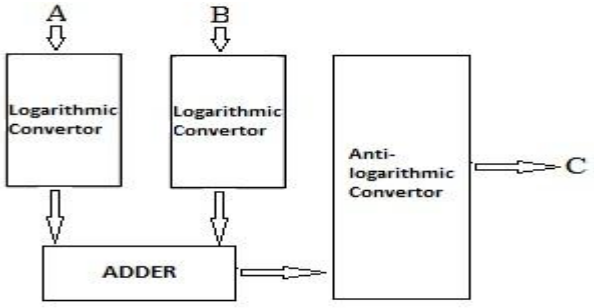


Figure -1: Logarithmic system block diagram

III. MITCHELL'S ALGORITHM

A logarithmic number system (LNS) [1] is introduced to simplify multiplication, especially in cases when the accuracy requirements are not rigorous. In LNS two operands are multiplied by finding their logarithms, adding them, and after that looking for the antilogarithm of the sum. One of the most significant multiplication methods in LNS is Mitchell's algorithm [1]. An approximation of the logarithm and the antilogarithm is essential, and it is derived from a binary representation of the numbers (2.1). The logarithm of the product is

$$\log_2(N_1 \cdot N_2) = K_1 + K_2 + \log_2(1+X_1) + \log_2(1+X_2) \qquad (2.1)$$

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:

$$\log_2(N_1 \cdot N_2) \approx K_1 + K_2 + X_1 + X_2 \qquad (2.2)$$

The characteristic numbers K_1 and K_2 represent the places of the most significant operands' bits with the value of '1'. For 16-bit numbers, the range for characteristic numbers is from 0 to 15. The fractions X_1 and X_2 are in range [0, 1).

The final MA approximation for the multiplication (where $P_{TRUE}=N_1 \cdot N_2$) depends on the carry bit from the sum of the mantissas and is given by:

$$P_{MA} = (N_1 \cdot N_2) = \begin{cases} 2^{(K_1+K_2)}(1+X_1+X_2), & X_1+X_2 < 1 \\ 2^{(K_1+K_2)}(X_1+X_2), & X_1+X_2 \geq 1 \end{cases} \qquad (2.3)$$

The final approximation for the product (2.3) 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 $2^{(K_1+K_2)}$ or $2^{(K_1+K_2+1)}$, depending on the X_1+X_2. If $(X_1+X_2) < 1$, 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. The reported MA-based multiplication is given in Algorithm 1.

Algorithm 1: (Mitchell's algorithm [1]).

1. *N_1, N_2: n-bits binary multiplicands, P_{MA}= 0:2n-bits approximate product*
2. *Calculate K_1: leading one position of N_1*
3. *Calculate K_2: leading one position of N_2*
4. *Calculate X_1: shift N_1 to the left by (n-K_1) bits*
5. *Calculate X_2: shift N_2 to the left by (n-K_2) bits*
6. *Calculate $K_{12}=K_1+K_2$*
7. *Calculate $X_{12}=X_1+X_2$*
8. *IF $X_{12} \geq 2n$ (i.e.$X_1+X_2 \geq 1$):*
   a. *Calculate $K_{12}=K_{12}+1$*
   b. *Decode K_{12} and insert X_{12} in that position of P_{approx}*

   else:
   a. *Decode K_{12} and insert '1' in that position of P_{approx}*
   b. *Append X_{12} immediately after this one in P_{approx}*
9. *Approximate $N_1 \cdot N_2=P_{MA}$*

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. Abed and Siferd [11] derived correction equations with coefficients that are a power of two, reducing the error and keeping the simplicity of the solution. McLaren's method [3], 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 Maralinga and Rangantathan [4].

IV. ITERATIVE LOGARITHMIC BASED MULTIPLICATION

This method [2] simplifies logarithm approximation introduced in (2.3) and introduces an iterative algorithm with various possibilities for achieving the multiplication error as small as required and the possibility of achieving the exact result. By simplifying the logarithm approximation introduced in (2.3), the correction terms could be calculated almost immediately after the calculation of the approximate product has been started. In such away, the high level of parallelism can be achieved by the principle of pipelining, thus reducing the complexity of the logic required by (2.3) and increasing the speed of the multiplier with error correction circuits. Looking at the binary representation of the numbers in (2.1), we can derive a correct expression for the multiplication:

$$\begin{aligned} P_{true} &\equiv (N_1 \cdot N_2) \\ &= 2^{(K_1+K_2)} + (N_1 - 2^{K_1})2^{K_2} + (N_2 - 2^{K_2})2^{K_1} + (N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2}) \end{aligned} \quad (2.5)$$

To avoid the approximation error, we have to take into account the next relation derived from (2.1):

$$X \cdot 2^k = N - 2^K \quad (2.6)$$

The combination of (2.5) and (2.6) gives:

$$\begin{aligned} P_{true} &\equiv (N_1 \cdot N_2) \\ &= 2^{(K_1+K_2)} + (N_1 - 2^{K_1})2^{K_2} + (N_2 - 2^{K_2})2^{K_1} + (N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2}) \end{aligned} \quad (2.7)$$

Let

$$P_{approx}^{(0)} \equiv 2^{(K_1+K_2)} + (N_1 - 2^{K_1})2^{K_2} + (N_2 - 2^{K_2})2^{K_1} \quad (2.8)$$

be the first approximation of the product. It is evident that

$$P_{true} = P_{approx}^{(0)} + (N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2}) \quad (2.9)$$

This method is very similar to Mitchell's Algorithm. The error is caused by ignoring the second term in (2.9). The term $(N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2})$ requires multiplication. If we discard it from (2.9), we have the approximate multiplication that requires only few shifts and add operations. Computational equation to MA multiplier (2.3) requires the comparison of the addend X_1+X_2 with 1. Instead of ignoring it and instead of approximating the product as proposed in (2.3), we can calculate the product $(N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2})$ in the same way as $P_{approx}^{(0)}$ and repeat the procedure until exact result is obtained. The evident difference between the proposed method and the method proposed by Mitchell is that the proposed method avoids the comparison of the addend X_1+X_2 with 1. In such a way, the error correction can start immediately after removing the leading ones form the both input operands N_1 and N_2. This is a key factor that allows further pipelining and reduces the required gates as we will

show lately. For this reason, an iterative calculation of the correction terms is proposed, as follows. The absolute error after the first approximation is

$$E^{(0)} = P_{true} - P^{(0)}_{approx} = (N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2}) \quad (2.10)$$

Note that $E^{(0)} \geq 0$. The two multiplicands in equation (2.10) are binary numbers that can be obtained simply by removing the leading '1' in the numbers N_1 and N_2 so we can repeat the proposed multiplication procedure with these new multiplicands:

$$E^{(0)} = C^{(1)} + E^{(1)} \quad (2.11)$$

Where $C^{(1)}$ is the approximate value of $E^{(0)}$ and $E^{(1)}$ is an absolute error when approximating $E^{(0)}$. The combination of (2.9) and (2.11) gives

$$P_{true} = P^{(0)}_{approx} + C^{(1)} + E^{(1)} \quad (2.12)$$

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.

$$P^{(1)}_{approx} = P^{(0)}_{approx} + C^{(1)} \quad (2.13)$$

If we repeat this multiplication procedure with i correction terms, we can approximate the product a

$$P^{(i)}_{approx} = P^{(0)}_{approx} + C^{(1)} + C^{(2)} + ... + C^{(i)} = P^{(0)}_{approx} + \sum_{j=1}^{i} C^{(j)} \quad (2.14)$$

The procedure can be repeated, achieving an error as small as necessary, or until at least one of the residues becomes a zero. Then the final result is exact: $P_{approx} = P_{true.}$ The number of iterations required for an exact result is equal to the number of bits with the value of '1' in the operand with the smaller number of bits with the value of '1'.

4.1. Reported algorithm for iterative logarithmic multiplier

The iterative MA-based multiplication [2] is given in Algorithm 2.

Algorithm 2: Iterative logarithmic multiplication based algorithm [2] with i correction terms

1. *N_1, N_2:n-bits binary multiplicands, $P^{(0)}_{approx}$=0:2n-bits first approximation, C(i)=0: 2n-bitsi correction terms, P_{approx}=0: 2n-bits product*
2. *Calculate K_1: leading one position of N_1*
3. *Calculate K_2: leading one position of N_2*
4. *Calculate $(N_1 - 2^{K_1})2^{K_2}$: shift $(N_1 - 2^{K_1})$ to the left by K_2 bits*
5. *Calculate $(N_2 - 2^{K_2})2^{K_1}$ shift $(N_2 - 2^{K_2})$ to the left by K_1 bits*
6. *Calculate $K_{12}=K_1+K_2$*
7. *Calculate $2^{(K_1+K_2)}$: decode K_{12}*
8. *Calculate $P^{(0)}_{approx}$: add $2^{(K_1+K_2)}$, $(N_1 - 2^{K_1})2^{K_2}$ and $(N_2 - 2^{K_2})2^{K_1}$.*
9. *Repeat i-times or until N_1=0 or N_2=0:*
   (a) *Set :$N_1 = (N_1 - 2^{K_1})$, $N_2 = (N_2 - 2^{K_2})$*
   (b) *Calculate K_1: leading one position of N_1*
   (c) *Calculate K_2: leading one position of N_2*
   (d) *Calculate $(N_1 - 2^{K_1})2^{K_2}$: shift $(N_1 - 2^{K_1})$ to the left by K_2 bits*
   (e) *Calculate $(N_2 - 2^{K_2})2^{K_1}$: shift $(N_2 - 2^{K_2})$ to the left by K_1 bits*
   (f) *Calculate $K_{12}=K_1+K_2$*
   (g) *Calculate $2^{(K_1+K_2)}$: decode K_{12}*
   (h) *Calculate $C^{(i)}$: add $2^{(K_1+K_2)}$, $(N_1 - 2^{K_1})2^{K_2}$ and $(N_2 - 2^{K_2})2^{K_1}$*
10. *$P^{(i)}_{approx} = P^{(0)}_{approx} + \sum iC^{(i)}$*

One of the advantages of the solution [2] is the possibility to achieve an arbitrary accuracy by selecting the number of iterations, i.e., the number of additional correction circuits as shown in Fig-3, but more important is that the calculation of the correction terms can start immediately after removing the leading ones from the original operands, because there is no comparison of the sum of the mantissas with 1.

The pipelined implementation of the basic block is shown in Fig. 2 and has four stages.

The stage 1 calculates the two characteristic numbers K_1, K_2 and the two residues $(N_1 - 2^{K_1}), (N_2 - 2^{K_2})$.

The residues are outputted **in stage 2**, which also calculates K_1+K_2, $(N_1 - 2^{K_1})2^{K_2}$ and $(N_2 - 2^{K_2})2^{K_1}$

The stage 3 calculates two terms $2^{(K_1+K_2)}$ and $(N_1 - 2^{K_1})2^{K_2} + (N_2 - 2^{K_2})2^{K_1}$

The stage 4 calculates the approximation of the product $P_{approx}^{(0)}$.

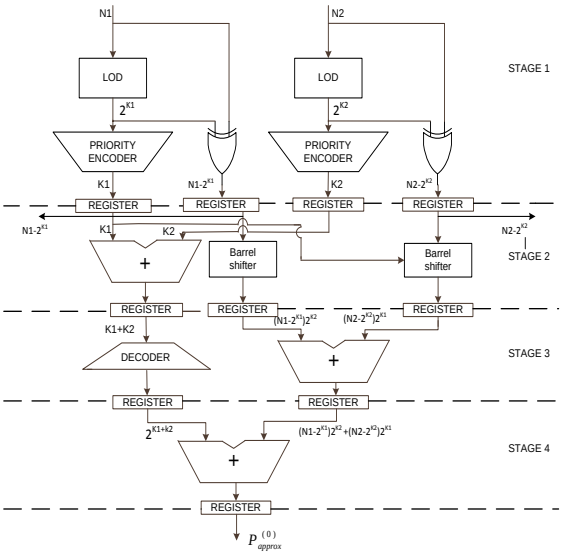


Figure -2: Pipelined basic block of iterative logarithmic multiplier

The pipelined multiplier with one, two and three correction circuits is presented in Fig. 3. The multiplier is composed of the three pipelined basic blocks, of which the first one calculates an approximate product $P_{approx}^{(0)}$, while the second and the third ones calculate the error-correction terms $C^{(1)}$ and $C^{(2)}$, respectively. The initial latency of the pipelined multiplier with two correction circuits is 6 clock periods, but after the initial latency, the products are calculated in each clock period. There are three multipliers implemented: with one error-correction circuit as shown in Fig-3(a), with two error-correction circuits as shown in Fig-3(b) and with three error correction circuits as shown in Fig-3(c). Each correction circuit is implemented as a basic block and is used to approximate the product according to (2.14).

4.2. **Error analysis of iterative logarithmic multiplier**

It is shown [2] that the maximum relative error decreases exponentially with a ratio of at least 2^{-2i}, and it reaches 0 when one of the multiplicands is 0. Table-1 presents the maximum relative errors for different numbers of ECCs [2].

Table -1: Maximum relative errors per number of ECCs used

Maximum relative errors per number of ECCs used.	
ECCs	Er,max(%)
0	25
1	6.25
2	1.56
3	0.39
4	0.097
5	0.02425

Hence, by observing above table it is obvious that for achieving maximum accuracy we have to decrease relative error and that can be done by adding extra error correcting circuits (ECC's). So, Depending upon requirement of application choice of ECC's is done .but as we increase number of ECC's size of hardware increases that is accuracy and area (size) are directly proportional hence, As increase in accuracy corresponding increase in area is inevitable.

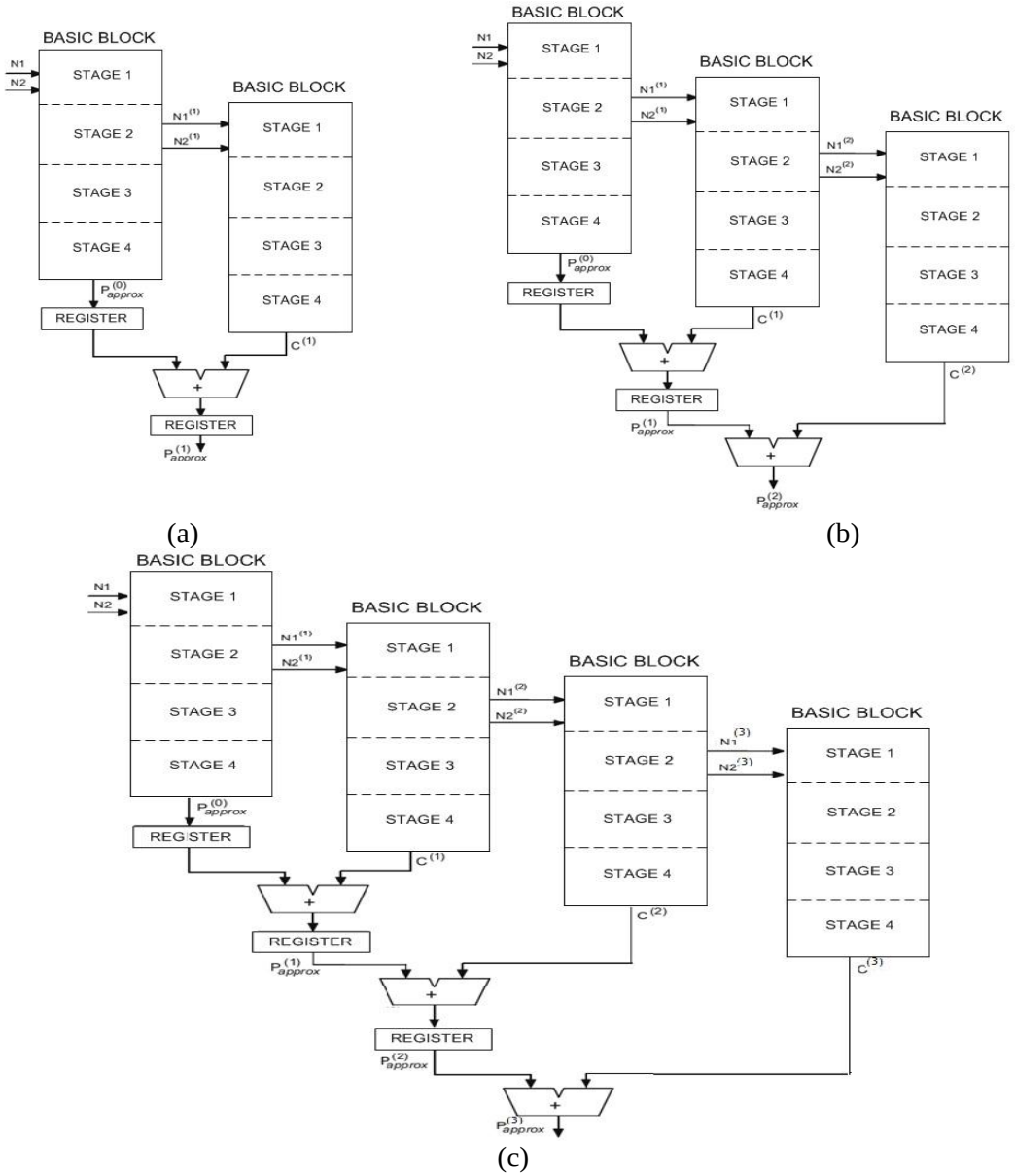


Figure -3: Pipelined iterative logarithmic multiplier (a) With one error correction circuit (b) With two error correction circuit (c) With three error correction circuit

4.3. **Limitation of iterative logarithmic multiplier**

As described in iterative logarithmic multiplier method[2] basic block is used for calculation of an approximate product $P_{approx}^{(0)}$, while for error correction circuits array of same basic block are used which calculate the error-correction terms $C^{(1)}$, $C^{(2)}$, ...$C^{(i)}$ each from ith block of error correction circuit depending upon accuracy required. As shown in error analysis [2] as requirement of accuracy increase no. of error correction circuit increases with in turn increase area and power consumption because each addition of error correction block consumes extra amount of power.

Thus, designing multiplier as per [2] for application which required more accuracy also required more area and more power consumption which leads to some constrains over implementation. Hence proposed solution overcomes this limitation by using recursive logic for error correction circuit which substitute array of basic block by only single basic block with additional circuit. In proposed solution we modified iterative logarithmic multiplier using recursive logic for low-power and area-efficient applications.

V. MODIFIED ALGORITHM FOR ITERATIVE LOGARITHMIC MULTIPLIER.

Modified Iterative logarithmic Multiplier with Recursive Logic is basically modified version of existing algorithm of iterative logarithmic multiplier [2] in which only step 2 is extra added, condition of step 10 is slightly modified and in last step addition is carried out in recursive manner. Algorithm is as follows:

Algorithm 3: Modified Iterative logarithmic Multiplier

1. *M_1, M_2:n-bits binary multiplicands, $P^{(0)}_{approx}$=0:2n-bits first approximation, $C^{(i)}$=0: 2n-bits i correction terms, P_{result}=0: 2n-bits product*
2. *N_1, N_2:n-bits binary output of select logic Block*
   Set: $N_1=M_1$ or $(N_1-2^{K_1})$, $N_2=M_2$ or $(N_2-2^{K_2})$ Depending upon Select pin input
3. *Calculate K_1: leading one position of N_1*
4. *Calculate K_2: leading one position of N_2*
5. *Calculate $(N_1-2^{K_1})2^{K_2}$: shift $(N_1-2^{K_1})$ to the left by K_2 bits*
6. *Calculate $(N_2-2^{K_2})2^{K_1}$ shift $(N_2-2^{K_2})$ to the left by K_1 bits*
7. *Calculate $K_{12}=K_1+K_2$*
8. *Calculate $2^{(K_1+K_2)}$: decode K_{12}*
9. *Calculate $P^{(0)}_{approx}$: add $2^{(K_1+K_2)}$, $(N_1-2^{K_1})2^{K_2}$ and $(N_2-2^{K_2})2^{K_1}$.*
10. *Repeat until $N_1=0$ or $N_2=0$:*
    (a) Set :$N_1=(N_1-2^{K_1})$, $N_2=(N_2-2^{K_2})$
    (b) Calculate K_1: leading one position of N_1
    (c) Calculate K_2: leading one position of N_2
    (d) Calculate $(N_1-2^{K_1})2^{K_2}$: shift $(N_1-2^{K_1})$ to the left by K_2 bits
    (e) Calculate $(N_2-2^{K_2})2^{K_1}$: shift $(N_2-2^{K_2})$ to the left by K_1 bits
    (f) Calculate $K_{12}=K_1+K_2$
    (g) Calculate $2^{(K_1+K_2)}$: decode K_{12}
    (h) Calculate $C^{(i)}$: add $2^{(K_1+K_2)}$, $(N_1-2^{K_1})2^{K_2}$ and $(N_2-2^{K_2})2^{K_1}$
11. $P_{result}=P^{(0)}_{approx}+\sum_i C^{(i)}$ using Recursive Adder.

5.1. Hardware implementation of modified algorithm for iterative logarithmic multiplier

The proposed modified iterative logarithmic multiplier hardware implementation is basically modified version of existing iterative logarithmic multiplier's basic block [2].in modified version only single basic block is used instead of array of basic block that used in [2] this can be done using recursive logic .The detail description of Hardware implementation of proposed modified iterative logarithmic multiplier is as follows.

5.2. Implementation of select logic block

To implement Modified iterative logarithmic multiplier with only one basic block[2] for calculations of first approximate of product $P^{(0)}_{approx}$ and error correction terms $C^{(i)}$ to be added in approximate value in successive cycles to get exact product P_{true} select logic block is necessary. We use two Select logic block each one has two 16 bit input operand, one select line, one status signal output which indicate one of the operand is zero this is important because it indicate no further error correction require and observing this signal we can load our next operands and other 16 bit output which is selection between two input operand depending upon select line data this output then work as input to one of basic block LOD.

Suppose, M_1, M_2: n-bits binary multiplicands, $N_1^{(i)}, N_2^{(i)}$:n-bits binary output of select logic Block after i^{th} iterations $(N_1^{(i)}-2^{K_1^{(i)}}), (N_2^{(i)}-2^{K_2^{(i)}})$:n-bits binary residue of basic block after i^{th} iteration work as input

Set: $N_1^{(i)}=M_1$ or $(N_1^{(i-1)}-2^{K_1^{(i-1)}})$, $N_2^{(i)}=M_2$ or $(N_2^{(i-1)}-2^{K_2^{(i-1)}})$ Depending upon Select line input. Initially select pin input is high and there are no residue terms which makes $N_1^{(0)}=M_1$ and $N_2^{(0)}=M_2$ and then until either of residue become zero it assign $N_1^{(i)}=(N_1^{(i-1)}-2^{K_1^{(i-1)}})$ and $N_2^{(i)}=(N_2^{(i-1)}-2^{K_2^{(i-1)}})$ .after becoming either of residue zero it waits till next operands are not loaded.

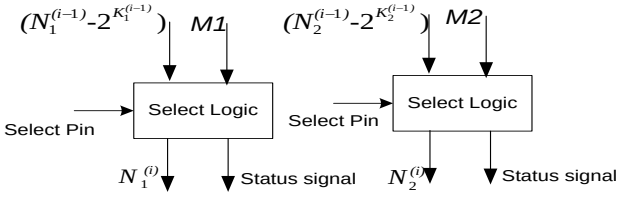


Figure -4: Block Diagram of select logic

5.3. **Implementation of Basic Block (BB)**

A basic block (BB) is the existing multiplier [2] with no correction terms. The task of the basic block is to calculate one approximate product according to (2.8). This basic block consists of two 16-bit leading-one detectors (LODs), two encoders, two 32-bit barrel shifters, a decoder unit and two 32-bit adders. Two input operands are given to the LODs and the encoders. The LOD units are used to remove the leading one from the operands, which are then passed to the barrel shifters. The LOD units also include zero detectors, which are used to detect the zero operands. The LOD units and the zero detectors are implemented as in [1], while the barrel shifters are used to shift the residues according to (2.8). The decoder unit decodes K_1+K_2, i.e. it puts the leading one in the product. The leading one and the two shifted residues are then added to form the approximate product. The basic block is then used in subsequent implementations to implement correction circuits.

5.4. **Implementation with recursive logic for error correction**

To increase the accuracy of the multiplier, we implemented multipliers with error-correction circuits (ECC). The error correction circuit is used to calculate the term $C^{(1)}$ in (2.12) and thus approximates the term $(N_1 - 2^{K_1}) \cdot (N_2 - 2^{K_2})$ in (2.7).To implement the proposed multiplier, we used the recursive logic along with basic block[2]. A block diagram of the proposed logarithmic multiplier with recursive logic circuit is shown in Fig-5.

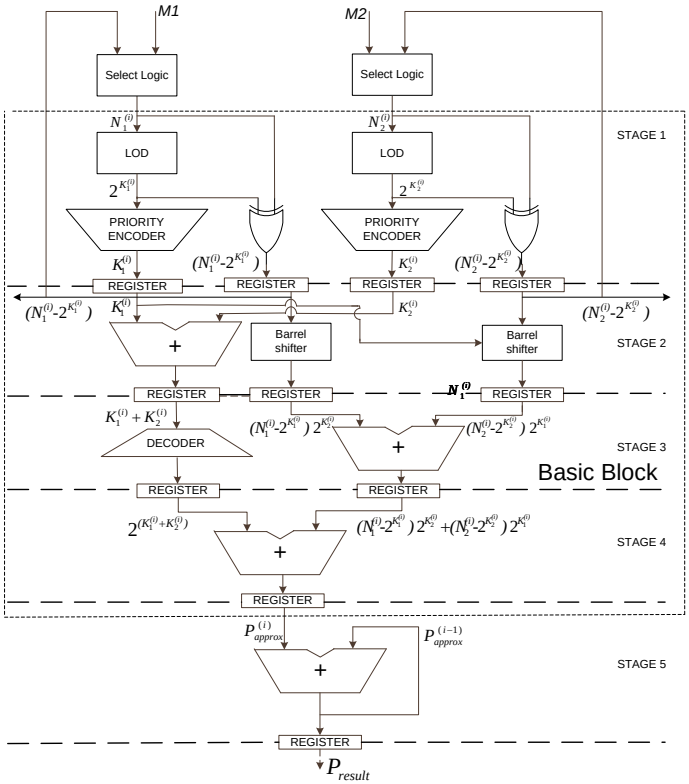


Figure -5: Block Diagram of Modified Pipelined iterative logarithmic multiplier

The proposed multiplier is composed of basic block [2], which initially calculates the first approximation of the product $P^{(0)}_{approx}$, while in next cycle it calculates the error correction terms $C^{(i)}$ with the help of residue and Using recursive and select logic. This error correction terms are then added with first approximate of product $P^{(0)}_{approx}$ till either of residue become zero. For adding we use recursive adder circuit. Hence, we implemented error correction circuit using same basic block and is used to approximate the product according to (2.14).

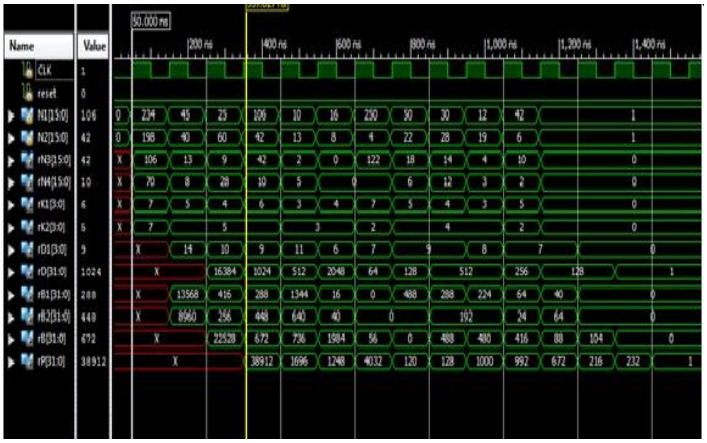
The pipelined multiplier with recursive logic for error correction is presented in Fig-5. The initial latency of the pipelined multiplier with recursive logic for error correction is 4 clock cycles for 1st approximate product $P^{(0)}_{approx}$, but after the initial latency, the approximate products $P^{(i)}_{approx}$ are calculated in every clock cycle but due to single block for error correction calculation overall clock cycles required to get final result P_{result} is depends upon how many times error correction required means after how many iteration either of residue become zero. Hence, total period vary from minimum 1 clock cycle to maximum 7 clock cycles. as delay factor

is variable and depends on how many iteration required overall delay is not much affected by our modification but power consumption and area decreases considerably.

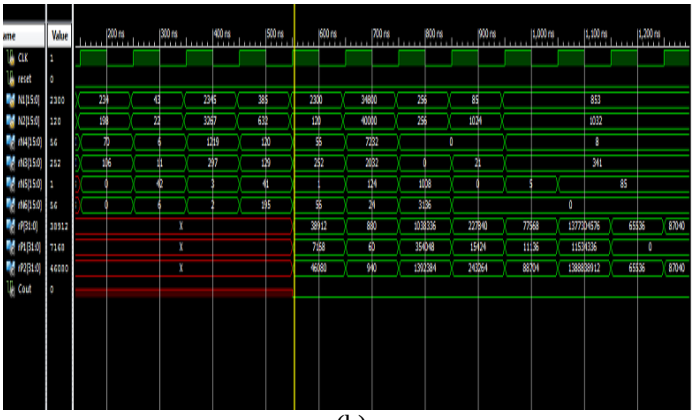
VI. SIMULATION RESULTS

This covers the results and simulation from synthesizing the 16-bit pipelined iterative logarithmic multiplier [2] with no error-correction terms, three multipliers with one, two and three error-correction terms and proposed modified iterative multiplier as shown in Fig-6. The tool used for the simulation and comparison of result was XILINX ISE 14.3. The Hardware implementation has been done on Xilinx xc3s1500-5fg676 FPGA.

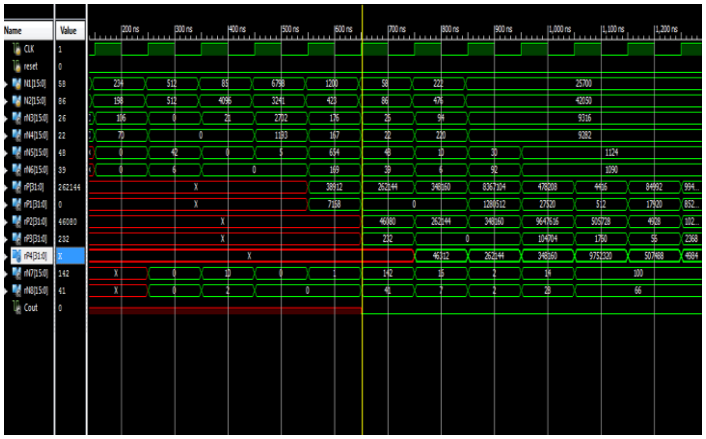
The complete hardware coding has been done in Verilog HDL. For power and timing analysis, we have used the XPower and Timing Analyzer tool of Xilinx.



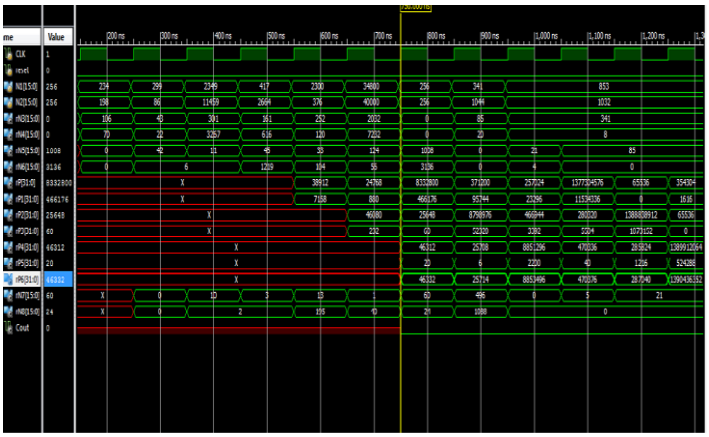
(a)



(b)



(c)



(d)

(e)

Figure -6: Simulation waveform of (a) Pipelined iterative logarithmic multiplier with no ECC (b)With one ECC (c) With two ECC (d) With three ECC (e) Modified Pipelined iterative logarithmic multiplier

VII. COMPARATIVE EVALUATION

In order to evaluate performance of the proposed multiplier and compare it with reported multiplier, we consider following performance parameters which gives us idea about which multiplier is comparatively better than other with respect to area, speed ,power and accuracy.

7.1. Device utilization of implementations

The device utilization expressed in terms of no. of basic building blocks utilized by circuit implemented. The basic building blocks utilization is expressed in terms of the number of slices, the number of 4-in-put look-up tables (LUTs) and the number of input–output blocks (IOBs). This ultimately gives how much area it occupies on the chip. Device utilization for all implemented multipliers is given in Table-2.

This shows that modified multiplier utilize slightly more device than basic block [2] but it occupy considerably less device than that of with error-correction circuits (ECC). Hence, we can say that modified multiplier is area-efficient.

Table -2: Device utilization of implemented multipliers

Multiplier	4-input LUTs	Slices	Slice FFs	IOBs
BB	424	219	198	67
BB + 1 ECC	857	454	423	67
BB + 2 ECC	1255	657	643	67
BB + 3 ECC	1649	858	858	67
Modified Multiplier	463	238	231	67

7.2. Estimated Power Consumptions of implementations

The power consumption is estimated at a clock frequency of 25 MHz with a signal (toggle) rate of 12.5%. With the Xilinx XPower Analyzer we have estimated the three main power components: Quiescent power, logic and signals power and the IOBs power. Quiescent power (also referred to as leakage) is the power consumed by the FPGA powered on with no signals

The estimated power consumptions for the implementations are given in Table-3.

Table -3: Maximum operating frequency of implemented multipliers

Multiplier	Logic and signals (mW)	IO Blocks (mW)	Quiescent (mW)	Total (mW)
BB	3.74	34.32	150.26	188.32
BB + 1 ECC	5.97	34.35	150.45	190.77
BB + 2 ECC	8.66	34.67	150.86	194.19
BB + 3 ECC	13.07	34.95	151.12	199.14
Modified Multiplier	4.22	34.43	150.42	189.07

This shows that power consume by proposed modified multiplier is less than that of basic block with one error correction circuits (ECC). Hence, proposed modified multiplier is also low-power consuming.

7.3. **Maximum Operating Frequency of implementations**

Maximum Operating Frequency is very important parameter in design because speed of operation is depends upon it. In Xilinx to get it we have to use timing Analyzer which will give minimum period through all possible path and also provide maximum operating frequency. Maximum Operating Frequency for all implemented multipliers are given in Table-4.

This shows that due to addition of extra select logic block in basic block which will increase the delay factor and reduce operating frequency to 146.434 MHz from 179.921MHz. As we know that there is always tradeoff between delay and area. Hence, as area reduces the delay is going to increase.

Table -4: Maximum operating frequency of implemented multipliers

Multiplier	Max. frequency(MHz)
BB	179.921
BB + 1 ECC	179.921
BB + 2 ECC	179.921
BB + 3 ECC	179.921
Modified Multiplier	146.434

7.4. **Maximum Relative errors of implementations**

A maximum Relative error is very important parameter in design because accuracy of operation is depends upon it. To calculate it we used expression of error analysis [2].
Maximum Relative error for all implemented multipliers are given in Table-5

Table -5: Maximum relative errors of implemented multipliers

Multiplier	Maximum relative errors Er,max(%)
BB	25
BB + 1 ECC	6.25
BB + 2 ECC	1.56
BB + 3 ECC	0.39
Modified Multiplier	0

This shows that as number of ECC's increases max .relative error decreases hence accuracy increases. but in modified multiplier utilize recursive logic because of that error-correction circuits (ECC) can be recursively used till one of operand becomes zero and finally gives accurate result. Hence, we can say that modified multiplier having zero relative error and result is accurate.

VIII. CONCLUSION

One of the most significant multiplication methods in logarithmic number system is Mitchell's algorithm [1] which is revolutionary in multiplier design. Based on this many improvement are proposed [2-7] among them Iterative logarithmic multiplier algorithm [2] uses same number system but for error correction it uses iterative method which has limitation such as with increase in accuracy power consumption and area will increases. To overcome it Modified Iterative logarithmic multiplier algorithm is proposed.

In this, we have investigated and proposed a new approach which requires less logic resources for its implementation i.e. efficient and this can achieve maximum accuracy and this is useful in low-power consumption and area-efficient applications. We have shown that the calculation of the correction terms can be

performed using single block with the help of recursive logic. After the initial latency, the approximate products $P_{approx}^{(0)}$ are calculated in every clock cycle but due to single basic block for error correction calculation overall clock cycles required to get final result P_{result} is depends upon how many times error correction required means after how many iteration either of residue become zero. Hence, total period to get final result vary from minimum 1 clock cycle to maximum 7 clock cycles. This shows that for large amount data delay is balanced and overall delay minimized.

The comparative evolution of proposed approach shows it improves the accuracy, area efficiency and consumes significantly less power in expense of slight increase in delay due to addition of extra select logic block in basic block [2] which will reduced maximum operating frequency to 146.434 MHz from 179.921MHz. Hence, proposed modified iterative logarithmic multiplier has been proved good for low-power and area-efficient applications

REFERENCES

Journal Papers:

[1] J.N. Mitchell, Computer multiplication and division using binary logarithms, IRE Transactions on Electronic Computers EC-11 (1962) 512–517.

[2] Z. Babić, A. Avramović, P. Bulić. An iterative logarithmic multiplier Microprocessors and Microsystems, Volume 35, Issue 1, February 2011, Pages 23-33

[3] D.J. McLaren, Improved Mitchell-based logarithmic multiplier for low-power DSP applications, in: Proceedings of IEEE International SOC Conference 2003, 17–20 September 2003, pp. 53–56.

[4] V. Mahalingam, N. Rangantathan, Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition, IEEE Transactions on Computers 55 (2) (2006) 1523–1535.

[5] E.L. Hall, D.D. Lynch, S.J. Dwyer III, Generation of products and quotients using approximate binary logarithms for digital filtering applications, IEEE Transactions on Computers C-19 (2) (1970) 97–105.

[6] M-H. Jing, Z-H. Chen, J-H. Chen, Y-H. Chen, Reconfigurable system for high-speed and diversified AES using FPGA, Microprocessors and Microsystems 31(2) (2007) 94–102.

[7] J.A. Kalomiros, J. Lygouras, Design and evaluation of a hardware/software FPGA-based system for fast image processing, Microprocessors and Microsystems 32 (2) (2008) 95–106.

[8] M.H. Rais, Efficient hardware realization of truncated multipliers using FPGA, International Journal of Applied Science 5 (2) (2009) 124–128.

[9] V. Hampel, P. Sobe, E. Maehle, Experiences with a FPGA-based reed/solomon-encoding coprocessor, Microprocessors and Microsystems 32 (5–6) (2008) 313–320.

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

[11] K.H. Abed, R.E. Sifred, VLSI implementation of a low-power antilogarithmic converter, IEEE Transactions on Computers 52 (9) (2003) 1221–1228.

Books:

[12] J.L. Hennessy, D.A. Patterson, computer architecture: a quantitative approach, fourth ed., Morgan Kauffman Pub., 2007.

[13] Verilog HDL: a guide to digital design and synthesis, second edition, sunsoft press, 1996 - Computers by Samir Palnitkar.

BIOGRAPHIES

Rohan Appasaheb Borgalli received his M.Tech degree in Digital Systems from Motilal Nehru National Institute of Technology (MNNIT), Allahabad. in 2013.His research interests include Digital Circuits and Systems Design, Digital Signal and Image Processing.

Hari Pratap Gautam received his M.Tech degree in Digital Systems from Motilal Nehru National Institute of Technology (MNNIT), Allahabad. in 2013.His research interests include VLSI Design and Signal Processing

Winner George Parayil received his M.Tech degree in Digital Systems from Motilal Nehru National Institute of Technology (MNNIT), Allahabad. in 2013. His research interests include wireless networks, gesture recognition and embedded systems.