Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, No. 2,JULY - DEC 2023

An Efficient Logarithmic Multiplier Using Iterative Mitchell's Algorithm using VLSI

K.Srikanth[1], M.Ganesh[2], T.Bhavani[3], S.Naresh[4],P.Sumana[5],K.Mallaiah[6],B.Ashwini [7]

[1,2,3,4,5,6,7] Assistant Professor, Department of ECE, Sri Indu Institute of Engineering & Technology, Hyderabad

Abstract: ***Multiplication is a basic arithmetic operation. Multiplication operations such as Fast Fourier Transforms, Multiplication and accumulation units, Convolution are some of the computation-intensive arithmetic functions often encountered in Digital Signal processing applications. Generally, Logarithm based multipliers are used in these cases which introduce certain errors. These errors are approximated by various methods. In this paper a simple architecture of a 16X16 logarithm based multiplier is proposed which uses simple combinational and sequential circuits to obtain an exact product. The multiplier has an arbitrary execution time which varies from 0 clock cycles to 15 clock cycles (neglecting the combinational delay) and whose mean delay is 7.5 clock cycles. This architecture is designed and simulated in 'ModelSim' simulation tool.***

Keywords: logarithmic multiplication, mitchell's algorithm

1. Introduction

Logarithmic multiplication is the process which involves calculating the product of two operands by converting the operands into Logarithmic Number System. The procedure for calculating the product involves converting the operands into their respective logarithms, adding the logarithmic result and computing the anti- logarithm of that result. This procedure is simpler as the addition operation replaces product operation in Logarithmic Number Systems. However, this procedure introduces a setback as the logarithms and anti-logarithms cannot be computed exactly. So, these methods introduce errors as exact values of Logarithms and anti-logarithms cannot be obtained and one is obliged to approximate the results of Logarithms and anti-logarithms. Such a method is Mitchell's Algorithm based multiplier [1], which approximates $log_2(1+x)$ as x, where x represents mantissa of a number.. An iterative architecture similar to Mitchell's Algorithm based multiplier was proposed by Patricio Buli´c and his team [3] which models the true product as the sum of approximate product and error. The error here is in the form of the product of two new operands which can be again fed into a similar block and whose approximate product can be added to the previous result, so as to reduce the overall error. The overall error is not reduced through any direct approximation technique, which form the principal constituents in most of the logarithmic based multipliers, rather provides an iterative solution to reduce the error. Some of the direct error approximation techniques are segmentation and interpolation techniques [6]. The proposed architecture follows an algorithm which is similar to the iterative algorithm, and achieves an exact result. The rest of the paper is organized as follows. Section II is subdivided into 2 parts (A, B). Part A and B explain Mitchell's algorithm and iterative multiplication algorithm. Section III presents the proposed architecture which uses the modified iterative block to arrive at the exact results and also explains the functionality of each block used in the architecture in detail. Section IV provides simulation results. Section V draws conclusions.

2. Mitchell's Algorithm

Any binary integral number can be written as:

$$N = 2^k\left[1 + \sum_{n=0}^{k-1} 2^{n-k}.Zi\right] \quad (1)$$

Where 'k' is the position of the most significant bit whose value is '1'. 'Zi' is the value of the bit in the i^{th} position.

The above equation can be further modeled as:

$$N = 2^k[1 + X] \quad (2)$$

Where *X* is the mantissa part

$$log_2 N = k + log_2(1 + x) \quad (3)$$

Mitchell's algorithm approximates $log_2(1+x)$ with x

So, for any two operands N1 and N2,

$$N_1 = 2^{k_1}[1 + X_1] \quad (4)$$

$$N_2 = 2^{k_2}[1 + X_2] \quad (5)$$

$$log_2 N_1 = k_1 + log_2(1 + x_1) \quad (6)$$

$$log_2 N_2 = k_2 + log_2(1 + x_2) \quad (7)$$

From Mitchell's approximation,

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, No. 2,JULY - DEC 2023

$log_2 N_1 = k_1 + x_1$ and $log_2 N_2 = k_2 + x_2$.

$log_2(N_1.N_2) = k_1 + x_1 + k_2 + x_2$. (8)

The error here is positive as $log_2(1+x)$ is always greater than or equal to x and the error ranges from 0-11% [2] . Various techniques were proposed to reduce this error, some of them being Operand Decomposition method [5], using look-up tables [4], Segmentation and interpolation methods [6]. Each method has its own tradeoffs between architecture complexity, accuracy and delay time and is generally used where certain errors are tolerable. Such conditions are often met in Digital Signal Processing applications.

a) Simple Iterative Logarithmic Multiplier

This method is similar to Mitchell's algorithm and uses an iterative method which gives a possibility to achieve an error as small as one desire and even might achieve an exact result.

b) Mathematics Involved:

$$N_1 . N_2 = 2^{k1}[1 + X_1]. 2^{k2}[1 + X_2] \quad (9)$$

$$N_1 . N_2 = 2^{k1+k2}. [1 + X_1 + X_2] + 2^{k1+k2}.[X_1. X_2] \quad (10)$$

$$N_1 . N_2 = 2^{k1+k2} + X_1. 2^{k1+k2} + X_2. 2^{k1+k2} + [X_1. X_2]. 2^{k1+k2} \quad (11)$$

From the initial equation, we can write $X. 2^k = (N - 2^k)$. Therefore

$X_1.2^{k1} = (N_1 - 2^{k1})$ and $X_2. 2^{k2} = (N_2 - 2^{k2})$ (12)

$$N_1 . N_2 = 2^{k1+k2} + (N_1 - 2^{k1}). 2^{k2} + (N_2 - 2^{k2}). 2^{k1} + (N_1 - 2^{k1}) . (N_2 - 2^{k2}) \quad (13)$$

$P_{true} = N_1 . N_2$

$P_{true} = P_{appx} + E$

Where P_{true} is the exact product, P_{appx} is the approximate productand 'E' is the error.

$$P_{appx} = 2^{k1+k2} + (N_1 - 2^{k1}).2^{k2} + (N_2 - 2^{k2}). 2^{k1} \quad (14)$$

$$E. = (N_1 - 2^{k1}) . (N_2 - 2^{k2}) \quad (15)$$

The error here is again a product of two operands for which the same arithmetic can be followed and subsequently adding the results will give the more accurate value of the product. On further following the same procedure repeatedly, the accurate product can be achieved at some point. The below block diagram is the architecture to achieve the above results. The architecture of a 16x16 bit iterative block uses Leading One Detectors (16 bit), Priority encoders (16 x 4), Barrel Shifters (32 bit), Ripple Carry Adders (4 bit and 32 bit), Decoders (5 x 32) and XOR banks (16 bit).

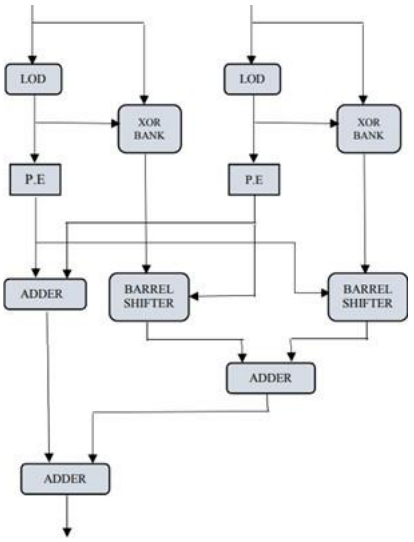

Figure 1: Architecture of the iterative block.

c) Algorithm:

Step 1: Inputs are given to the Leading One Detectors (LOD's) Outputs of which will be 2^k1 and 2^k2 .

Step 2: With inputs as 2^k1 and 2^k2 , priority encoders computethe values of k1 and k2.

Step 3: $(N_1 - 2^k1)$ and $(N_2 - 2^k2)$ are the outputs of XOR banks whose inputs are the Operands and outputs from Leading one detectors.

Step 4: Barrel shifters compute the values of $(N_1 - 2^{k1}). 2^{k2}$ and $(N_2 - 2^{k_2}). 2^{k_1}$.

Execution time = (number of iterations – 1)*(clock period) (17)

Step 5: The results of the two barrel shifters are added to obtain the sum $(N_1 - 2^{k1}). 2^{k2} + (N_2 - 2^{k2}). 2^{k1}$.

Step 6: The values of k_1 and k_2 obtained in step 2. are added and the result is given as an input to a Decoder, which gives the value of 2^{k1+k2} as output.

Step 7: The results obtained in step 5. and step 6. are added to give $2^{k1+k2} + (N_1 - 2^{k1}). 2^{k2} + (N_2 - 2^{k2}). 2^{k1}$ as output.

Step 8: The error operands are the outputs of the XOR banks.

3. Proposed Method

We have to note that each error operand in the above algorithm is the result of removing the most significant bit with the value '1' from the input operands to the iterative block. So by successive iterations (and adding the

approximate products of each iteration) at least one of the error operands becomes '0' at some point, which means that the error is '0' at that point, and the accurate product is obtained. The question here is, how to detect that instance when the accurate product is achieved.

For explaining this more clearly we shall take an example, here we will concentrate only on the errors after each iteration.

Let $N_1 = 1001010011$ *and* $N_2 = 10000101$

We know that error = $(N_1 - 2_{k1}).(N_2 - 2_{k2})$

We should note that $(N_1 - 2_{k1})$ is the value after removing MSB from the operand.

Therefore after 1st iteration, the error will be equal to $(1010011).(101)$

After 2nd iteration, the error will be equal to $(10011).(1)$

After 3rd iteration, an error will be equal to $(11).(0)$ which is '0' that means, at this instant accurate product is obtained.

Therefore we can state that the number of iterations to obtain an accurate product is equal to the minimum of the number of 1's in the two operands

Let n_1 and n_2 be the number of 1's in the input operands N_1 and N_2. Then,

$$number\ \text{of}\ iterations = \text{Min}\ [n1\ , n2\ ] \qquad (16)$$

Modified iterative block:

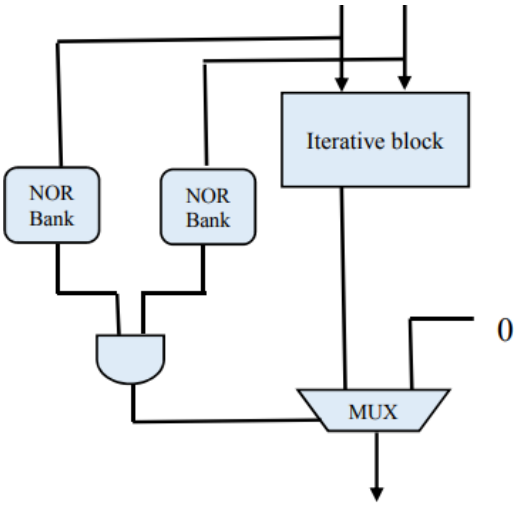

Figure 2: Modified iterative block

The Iterative block uses 16x4 priority encoders for which 16'b0 is an invalid input. So, the block does not perform as expected. So we modified the iterative block by including a combinational logic circuit which bypasses the case of the inputs being 16'b0, to act as expected (the iterative block is expected to give a 16'b0 as output even if any one of its operands is 16'b0).

Concept of the proposed architecture:

To address the question of detecting the instance at which the accurate product is obtained we use a check block which checks whether any of the error operands is '0'. Check block takes the error operands as input, gives a 'High' output when no operand is '0' and gives 'Low' output when any one of the operands is '0'.

We have to note that a transition from 'High' to 'Low' happens when the error becomes '0'. We use this condition to detect the instance at which error becomes zero or accurate product is achieved. The architecture should not allow new inputs or the initial inputs while the iterations are in progress, this is controlled by a control block which takes the input from the check block.

The control block allows the error operands as inputs to the iterative block when the output of the check block is 'High'. This block keeps on allowing the error operands until NOR Bank Iterative block NOR Bank MUX 0 the output of the check block is 'High'. The output of the check block being 'Low' means that the final result is achieved and there is no need of iterating the error operands further (since at least one of the error operands will be zero at this point and error becomes '0'). At this point, new inputs can be accepted. The control and check blocks are simple and can be constructed using logic gates, Multiplexers, and registers. These blocks are explained in detail in further discussions. A buffer register is used to store the temporary results of successive additions of products after each iteration.

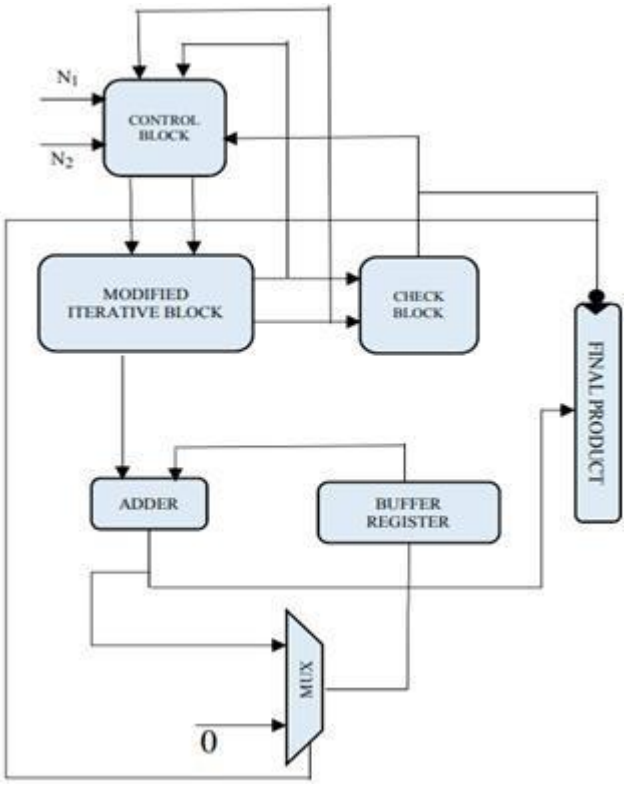

Figure 3: Block diagram of the Proposed Architecture

The register which stores the final product is driven by the output of the check block which serves as a negative edge

clock to the register. So the values in the register change only at the instances when the transition from 'High' to 'Low' occurs.

Check Block:

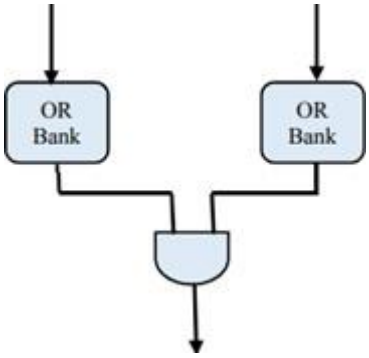

Figure 4: Block Diagram of the Check Block

As discussed earlier the check block takes error operands as inputs and gives a 'High' output when no operand is '0' and 'Low' output when any one of the operands is '0'. This is done by bit wise 'OR' operation of every digit of the operands individually and subsequently using an 'AND' gate. This check block is a typical zero detector.

Control Block:

N1	N2	Result	Expected result
5010 (20496)	3000 (12288)	0f030010 (251854864)	0f030000 (251854848)
76cd (30413)	0123 (291)	00870f01 (8851201)	00870b07 (8850183)
ffff (65535)	ffff (65535)	fffe0001 (4294836225)	fffe0001 (4294836225)
2301 (8961)	4fc6 (20422)	0ae064a0 (182477984)	0ae861c6 (183001542)
ffff (65535)	fffe (65534)	fffd0003 (4294770691)	fffd0002 (4294770690)
Fffd (65533)	ffff (65535)	fffc0004 (4294705156)	fffc0003 (4294705155)

5. Discussions

The architecture is tested on ModelSim simulation tool. As we have discussed earlier the delay is arbitrary and depends on the minimum of the number of 1's in both the operands, which can be observed from the simulation results. The wire 'w5' shows the delay caused in execution for different inputs. The mean delay is 7.5 clock cycles.

6. Conclusion

The architecture is tested on ModelSim simulation tool. As we have discussed earlier the delay is arbitrary and depends on the minimum of the number of 1's in both the operands,

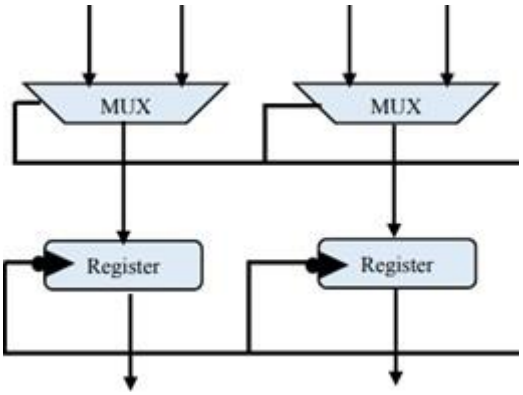

Figure 5: Block diagram of Control Block

The Control block controls the inputs to the modified iterative block. It does not allow new inputs or initial inputs while the iterations are in progress. The selection line in the above block is taken from the output of the check block. When the output of the check block is 'High' (happens when no error operands is '0') the selection line will be 'High' and allows the error operands for further iterations and blocks the initial inputs or new inputs. Selection line becomes '0' when any of the error operands is '0', then new inputs can be allowed.

4. Result Analysis

Figure 6: Simulation result

which can be observed from the simulation results. The wire 'w5' shows the delay caused in execution for different inputs. The mean delay is 7.5 clock cycles.

References

[1] Mitchell, J. N. (1962). Computer multiplication and division using binary logarithms. IRE Transactions on Electronic Computers, (4), 512-517.McLaren, D. J. (2003, September). Improved Mitchell-based logarithmic multiplier for low002Dpower DSP applications. In SOC Conference, 2003. Proceedings. IEEE International [Systems-on-Chip] (pp. 53-56). IEEE.

[2] McLaren, D. J. (2003, September). Improved Mitchell-based logarithmic multiplier for low002Dpower DSP applications. In SOC Conference, 2003. Proceedings. IEEE International [Systems-on-Chip] (pp. 53-56). IEEE.

[3] Agrawal, R. K., & Kittur, H. M. (2013, April). ASIC based logarithmic multiplier using iterative pipelined architecture. In Information & Communication Technologies (ICT), 2013 IEEE Conference on (pp. 362- 366). IEEE.

[4] Bulić, P., Babić, Z., & Avramović, A. (2010, October). A simple pipelined logarithmic multiplier. In Computer Design (ICCD), 2010 IEEE International Conference on

(pp. 235- 240). IEEE.

[5] Mahalingam, V., & Ranganathan, N. (2006, January). An efficient and accurate logarithmic multiplier based on operand decomposition. In VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on (pp. 6-pp). IEEE.

[6] Selina, R. R. (2013, April). VLSI implementation of piecewise approximated antilogarithmic converter. In Communications and Signal Processing (ICCSP), 2013 International Conference on (pp. 763-766). IEEE