International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 14, Number 3 (2021), pp. 249-255
© International Research Publication House. http://www.irphouse.com

# Optimised Logarithmic Multiplier Design with Predefined Iterations

**P Partha Koundinya[1], V Mani Deeepak[2] and Ch Jethin Sai[3]**

*[1, 2, 3] Electronics and Communication Engineering, SRM University-AP, India*

**Abstract**

Integrated circuits are used in various applications to perform arithmetic operations. Among various arithmetic operations, multiplication is one of the most important operations. The Iterative logarithmic multiplication is one of the multiplication techniques that provides accurate result. The iterative logarithmic multiplier is a circuit that consists of leading one detectors, barrel shifters, encoders, decoders and adders. As a part of this study, the existing architectures of few submodules have been studied. In addition, those architectures were integrated to build a logarithmic multiplier. The present work is focussed at designing a 16-bit iterative logarithmic multiplier only using combinational logic, determine the predefined iterations required to find the exact product of two numbers, and comparing the performance of such multipliers in terms of the power consumption and the look-up-tables (LUTs) through Verilog hardware description language in Xilinx vivado.

**Keywords** - Barrel shifter, Combinational logic, iterative logarithmic multiplier, predefined iterations, Verilog HDL.

## I. INTRODUCTION

There are many applications that need integrated circuits to perform arithmetic operations. Among various arithmetic operations, the present work is confined to the study of multiplication operation. Few methods of multiplication include non-iterative multiplication technique, iterative multiplication technique. The iterative multiplication technique can be used over the non-iterative multiplication technique to minimise the errors in the product [1].

In iterative multiplication algorithm, the product of any two numbers is calculated by the addition of approximate product and the error value [1]. The algorithm is briefly presented as follows:

Consider a number $N$. Now, N can be represented in binary form as:

$$N = 2^x\left(1 + \sum_{n=0}^{x-1} 2^{n-x} Z_n\right) = 2^x(1+m) \qquad \textbf{\textit{(1)}}$$

where x denotes position of MSB , n is the bit's position, $Z_n$ denotes the nth position bit and $m$ is the fraction.

Consider two numbers $N_1$ and $N_2$ and let their product be $P_{true}$.

Then,

$$\begin{aligned} P_{true} &= N_1.N_2 = 2^{x_1}(1+m_1).2^{x_2}(1+m_2) \\ &= 2^{x_1+x_2}(1+m_1+m_2) + 2^{x_1+x_2}(m_1 m_2) \end{aligned} \qquad \textbf{\textit{(2)}}$$

From (1),

$$N - 2^x = m.2^x \qquad \textbf{\textit{(3)}}$$

Substituing (3)in (2), we get,

$$P_{true} = 2^{x_1+x_2} + (N_1 - 2^{x_1})2^{x_2} + (N_2 - 2^{x_2})2^{x_1} + (N_1 - 2^{x_1})\,(N_2 - 2^{x_2}) \qquad \textbf{\textit{(4)}}$$

$$\text{Therefore,} \qquad P_{true} = P_{approx} + E$$

$$P_{approx} = 2^{x_1+x_2} + (N_1 - 2^{x_1})2^{x_2} + (N_2 - 2^{x_2})2^{x_1} \qquad \textbf{\textit{(5)}}$$

$$E = (N_1 - 2^{x_1})\,(N_2 - 2^{x_2}) \qquad \textbf{\textit{(6)}}$$

Below is the block diagram and brief explanation of the multiplication process with respect to the iterative multiplication algorithm [1]. It is also referred as basic block in [2].

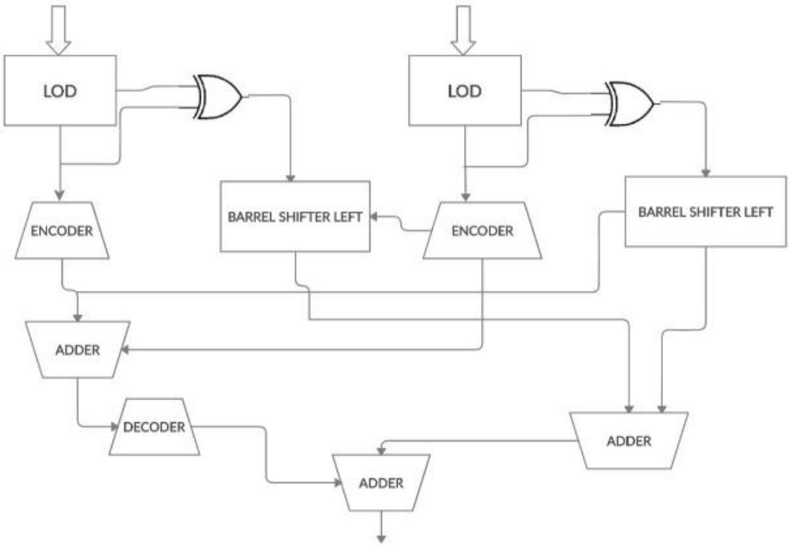


**Fig.1** Basic multiplier - block diagram

### I.I Barrel Shifter

A barrel shifter is a combinational circuit containing 2-1 multiplexers which shifts the data towards left by certain number of bits according to the selection input.

### I.II Leading one detector (LOD)

A leading one detector is a combinational circuit which is used to find the most significant bit (MSB) with value "1".

Let $N_1$, $N_2$ be two n-bit binary numbers and their product be a 2n-bit binary number, $P_{true}$. Then, the multiplication process is briefly explained [3], [5] as follows.

- The inputs to the LODs are $N_1$, $N_2$ and the corresponding outputs will be $2^{x_1}$ and $2^{x_1}$ ,where $x_1$ and $x_2$ represent the leading position of "1" in $N_1$ and $N_2$ towards the MSB side.
- The inputs to the encoders are $2^{x_1}$ and $2^{x_2}$ , whose outputs will be $x_1$ and $x_2$ respectively.
- With the input and output of LODs as inputs, the XOR gates give the outputs as $(N_1 - 2^{x_1})$ and $(N_2 - 2^{x_2})$.
- The barrel shifters are used to left-shift $(N_1 - 2^{x_1})$ by $x_2$ bits and $(N_2 - 2^{x_2})$ by $x_1$ bits. The corresponding outputs will be $(N_1 - 2^{x_1})2^{x_2}$ and $(N_2 - 2^{x_2})2^{x_1}$.
- A 32-bit adder is used to obtain the sum as $(N_1 - 2^{x_1})2^{x_2} + (N_2 - 2^{x_2})2^{x_1}$ by adding the outputs of the barrel shifters.
- Using a 4-bit adder, $x_1$ and $x_2$ are added. With this result as input, the decoder gives $2^{x_1+x_2}$ as the output.
- The outputs of the decoder and the 32-bit adder are given as inputs to another 32-bit adder and $2^{x_1+x_2} + (N_1 - 2^{x_1})2^{x_2} + (N_2 - 2^{x_2})2^{x_1}$ is obtained as a result.

Considering the outputs of XOR gates as the error terms, they are taken as inputs to the LODs and above process is repeated to obtain the product.

The value of $P_{approx}$ can be found in the first iteration of basic block, whereas successive iterations can be termed as error correction stages [2] which minimise the error in the product.

## II. METHODOLOGY

### II.I Designing a 16-bit Iterative Logarithmic Multiplier using combinational logic only

Initially, the modules of 16-bit LODs, 16-4 encoders, 32-bit barrel shifters, 5-32 decoder and 32-bit adder were designed using Verilog hardware description language in Xilinx Vivado (student version). All these modules were integrated to design the basic block module (shown in Fig.1) using structural modelling. Further, the logarithmic multiplier module was designed by creating several instances of the basic block module. This process is briefly mentioned below.

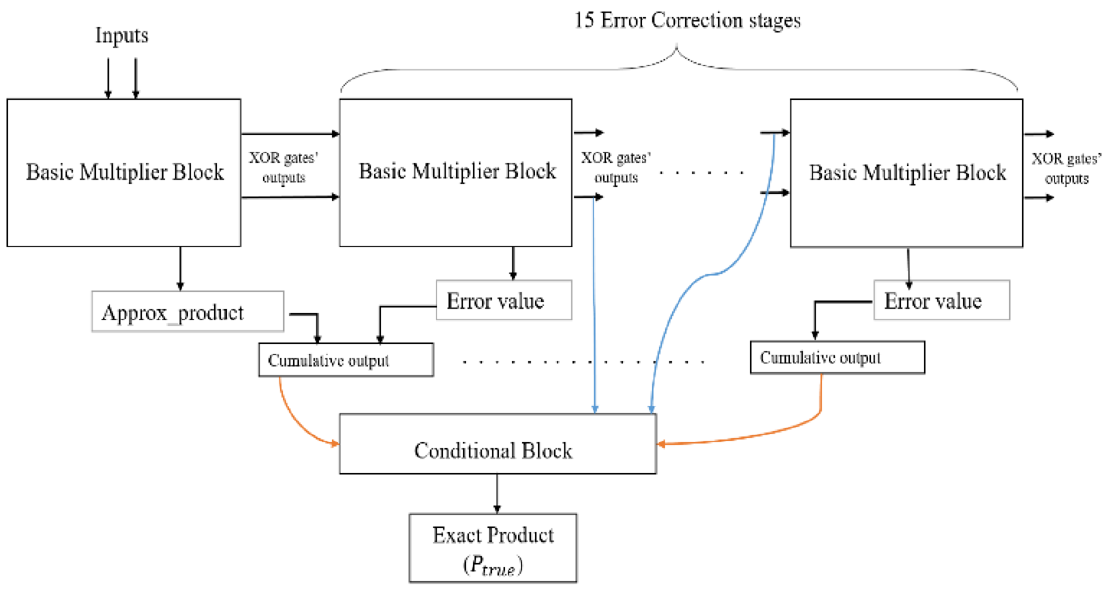


**Fig.2** Block diagram of proposed 16-bit multiplier

In the logarithmic multiplier module, the output of the first instance of the basic block is the approximate product value as shown in (5) for any two numbers. At this step, there might exist some error which can be calculated using (6) and this could be reduced by increasing the basic block iterations. To find the maximum number of basic block instances (iterations) for a 16-bit logarithmic multiplier, $(FFFF)_{16}$ was taken as input to the 2 LODs, because, it is the greatest 16-bit number. In this case, it was found that the exact product can be calculated with 16 basic block iterations which include a total of 15 error correction stages. Here, the input to the subsequent error correction blocks (shown in Fig. 2) were the outputs of the previous XOR gates. The values of the product and corresponding error terms has been presented in the results section.

However, for any two numbers less than $(FFFF)_{16}$, the exact product can be calculated with the number of basic block iterations less than or equal to 16. Mathematically, the value of error is equal to zero if any one of terms in (6) is zero and the required product can be found when the value of error is zero. So, by checking the outputs of the XOR gates of every stage with the help of a conditional block, the exact product can be obtained.

## II.II Designing logarithmic multipliers with variation in sub-module(s).

A 16-bit iterative logarithmic multiplier was designed as per the process explained above. The LODs and barrel shifters were designed as mentioned in [1]. Let this multiplier be named as “existing_design-1” for easier reference.

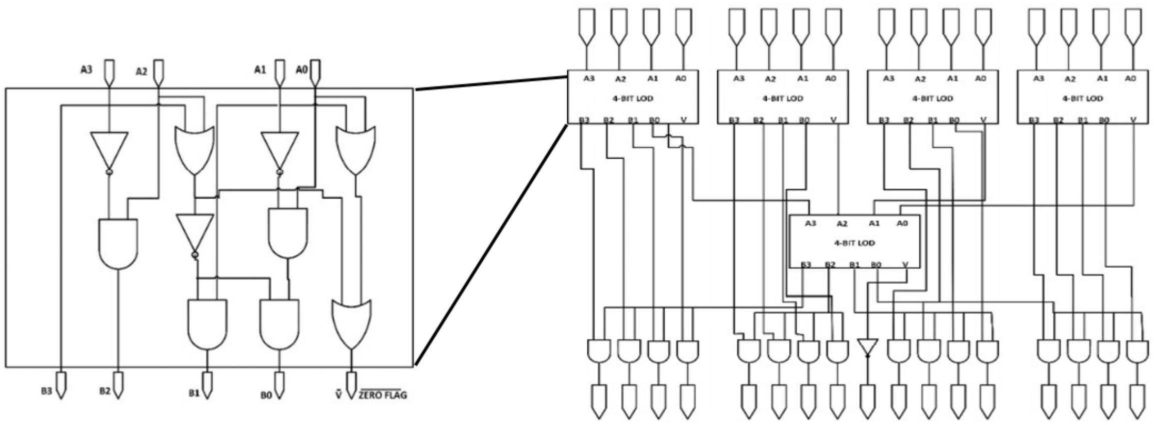


**Fig.3** 4-bit and 16-bit LODs respectively

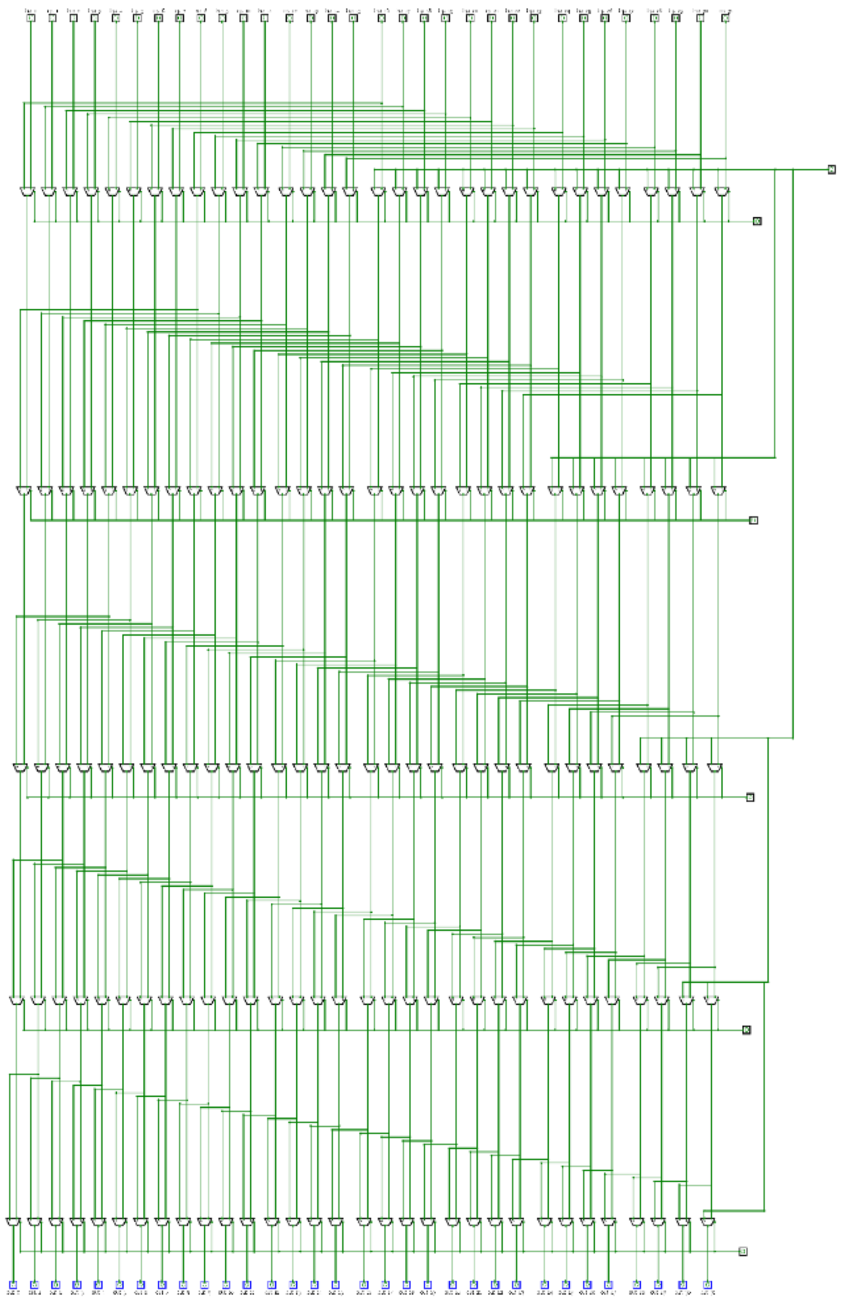


**Fig.4** 32-bit barrel shifter (used in existing_design-1)

Further, only the LOD and barrel shifter modules of "existing_design-1" were replaced with new modules (whose diagram is shown in below figures) to design another logarithmic multiplier ("existing design-2").

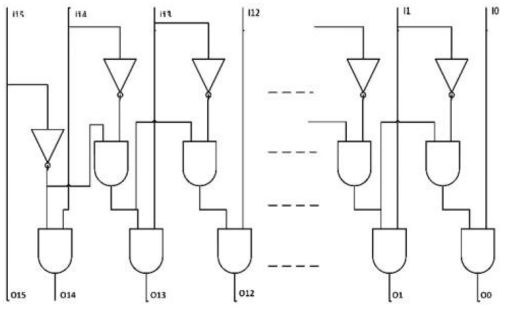


**Fig.5** 16-bit LOD (used in existing_design-2)

The barrel shifter module as proposed by [1] is a combination of ninety-eight 2-1 multiplexers and fifteen AND gates divided into four stages with four selection lines. While designing this module, ninety-eight instances of 2-1 multiplexers and fifteen instances of AND were created.

This new barrel shifter diagram is presented below.

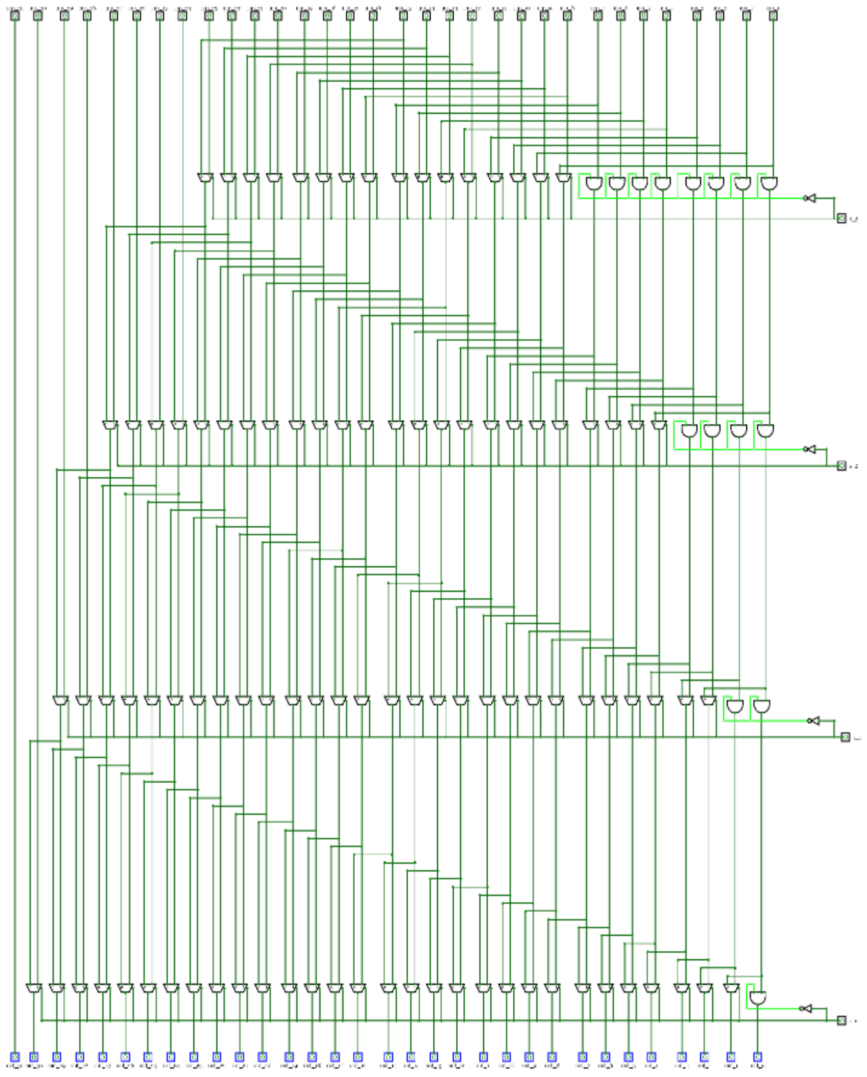


**Fig.6 32-bit barrel shifter (used in existing_design-2)**

## II.III Optimised barrel shifter design

To optimise the previous module of a 32-bit barrel shifter, the same block diagram shown in Fig.6 was followed. In this method, a module named "mux_row" was designed which is a combination of four 2-1 multiplexers.

Later, this module was repeatedly instantiated to utilise the multiplexers that are shown in Fig.6. This reduces the number of instances of the multiplexers.

Thus, the barrel shifter module in the multiplier "existing design-2" was replaced by the optimised barrel shifter to design a new 16-bit logarithmic multiplier named "new_design". A brief analysis of these multiplier designs is mentioned in Results section.

## III. RESULTS

The increase in the number of iterations of the basic block reduces the percentage of error in the product [2],[4],[6]. In the case of 4,8,16-bit logarithmic multipliers, the maximum number of iterations of basic block required to calculate the exact product is 4,8,16 respectively.

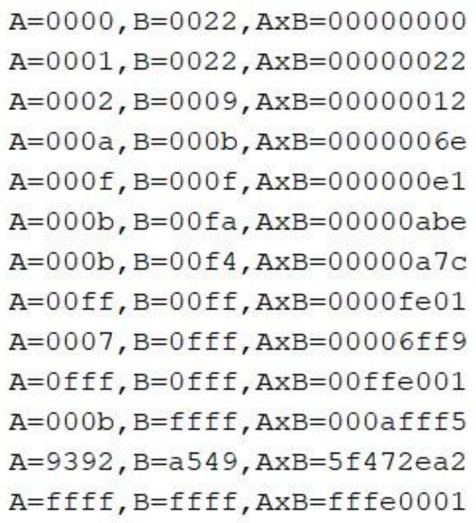
Hence, it is to interpret that, an n-bit iterative logarithmic multiplier may contain a maximum n iterations of the basic multiplier block. This implies, the number of error correction stages are n-1.

The power consumed by the digital logic of the optimised design of barrel shifter is nearly 0.85 % less than the barrel shifter design proposed by [1]. However, the overall power consumption of the logarithmic multiplier “new_design” is nearly 1.6 % higher than “existing_design-2”. This might be due to reusing the “mux_row” module in the “new_design” logarithmic multiplier.

**Table-I.** LUTs and Power Consumed by logic of all multiplier designs

| Logarithmic multiplier design | Number of LUTs | Power Consumed by logic |
|---|---|---|
| Existing_design-1 | 4172 | 42.830 W |
| Existing_design-2 | 3243 | 42.763 W |
| New_design | 3237 | 43.456 W |

The simulation result of the 16-bit iterative logarithmic multiplier is presented in below figure. The inputs and the output are in hexadecimal number system and this output remains same for all the three multiplier designs.



```
A=0000,B=0022,AxB=00000000
A=0001,B=0022,AxB=00000022
A=0002,B=0009,AxB=00000012
A=000a,B=000b,AxB=0000006e
A=000f,B=000f,AxB=000000e1
A=000b,B=00fa,AxB=00000abe
A=000b,B=00f4,AxB=00000a7c
A=00ff,B=00ff,AxB=0000fe01
A=0007,B=0fff,AxB=00006ff9
A=0fff,B=0fff,AxB=00ffe001
A=000b,B=ffff,AxB=000afff5
A=9392,B=a549,AxB=5f472ea2
A=ffff,B=ffff,AxB=fffe0001
```

**Fig.7** Simulation result

The value of the product and corresponding error for every stage of iteration, when the 2 inputs to the multiplier is $(FFFF)_{16}$ is presented below.

**Table-II.** Brief analysis of the product $(FFFF)_{16}$ x $(FFFF)_{16}$

| Block number | Input to the basic multiplier block (in Hexadecimal system) | | XOR gates’ outputs (in Hexadecimal system) | | Output of every stage (in Hexadecimal system) | Error value (in Hexadecimal system) | Cumulative product (in Hexadecimal system) |
|---|---|---|---|---|---|---|---|
| | A | B | $X_1$ | $X_2$ | | | |
| 1 | ffff | ffff | 7fff | 7fff | bfff0000 | 3fff0001 | bfff0000 |
| 2 | 7fff | 7fff | 3fff | 3fff | 2fff8000 | 0fff8001 | effe8000 |
| 3 | 3fff | 3fff | 1fff | 1fff | 0bffc000 | 03ffc001 | fbfe4000 |
| 4 | 1fff | 1fff | 0fff | 0fff | 02ffe000 | 00ffe001 | fefe2000 |
| 5 | 0fff | 0fff | 07ff | 07ff | 00bff000 | 003ff001 | ffbe1000 |
| 6 | 07ff | 07ff | 03ff | 03ff | 002ff800 | 000ff801 | ffee0800 |
| 7 | 03ff | 03ff | 01ff | 01ff | 000bfc00 | 0003fc01 | fffa0400 |
| 8 | 01ff | 01ff | 00ff | 00ff | 0002fe00 | 0000Fe01 | fffd0200 |
| 9 | 00ff | 00ff | 007f | 007f | 0000bf00 | 00003f01 | fffdc100 |
| 10 | 007f | 007f | 003f | 003f | 00002f80 | 00000F81 | fffdf080 |
| 11 | 003f | 003f | 001f | 001f | 00000bc0 | 000003c1 | fffdfc40 |
| 12 | 001f | 001f | 000f | 000f | 000002e0 | 000000E1 | fffdff20 |
| 13 | 000f | 000f | 0007 | 0007 | 000000b0 | 00000031 | fffdffd0 |
| 14 | 0007 | 0007 | 0003 | 0003 | 00000028 | 00000009 | fffdfff8 |
| 15 | 0003 | 0003 | 0001 | 0001 | 00000008 | 00000001 | fffe0000 |
| 16 | 0001 | 0001 | 0000 | 0000 | 00000001 | 00000000 | fffe0001 |

## IV. CONCLUSION

Thus, a 16-bit iterative logarithmic multiplier was designed, only using combinational logic, with predefined iterations of basic multiplier block and an optimised barrel shifter module.

## ACKNOWLEDGMENTS

We would like to express our deep gratitude to Prof. Siva Sankar Y for his generous support.

## REFERENCES

[1] Chaitanya MBK, Teja YS, Teja KR, Raghunath G , "An are efficient 16-bit logarithmic multiplier", *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN).*

[2] Agrawal RK, Kittur HM , "ASIC based logarithmic multiplier using iterative pipelined architecture", *Proceedings of 2013 IEEE Conference on Information and Communication Technologies (ICT 2013),* pp. 362-366.

[3] Pandit AA, Reddy Ch A, Narayan G, "Design and simulation of 16x16 bit iterative logarithmic multiplier for accurate results", *Proceedings of the 2nd international conference on Electronics, Communication and Aerospace Technology (ICECA 2018),* pp. 985-990.

[4] Babić Z, Avramović A, Bulić P, "A simple pipelined logarithmic multiplier", *2010 IEEE International Conference on Computer Design,* 2010.

[5] Mahalingam V, Ranganathan N, "An efficient and accurate logarithmic multiplier based on operand decomposition", 19th *International Conference on VLSI Design* held jointly with 5th *International Conference on Embedded Systems Design (VLSID06),* 2006.

[6] Ahmed S, Srinivas M, "An improved logarithmic multiplier for media processing", *Journal of Signal Processing Systems,* 2018.