# RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing

Manchiryala Manogna

December 25, 2020

# RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing (December2020)

***Manchiryala Manogna****, M.tech, Dept. of ECE, manchiryalamanogna369@gmail.com,mrce.in, PG Student of Malla Reddy College of Engineering, MRCE, Maisammaguda, Hyderabad.*

***Abstract*—In this paper, we propose an approximate multiplier that is high speed yet energy efficient. Now-a-days, Energy minimization is one of the main design requirements especially in the portable gadgets i.e., smart phones, tablets and so on. In these types of gadgets, DSP blocks are key components, where the computational core of these blocks is the arithmetic logic unit where multiplications have a greatest share. So, by the use of the multipliers the computational part of multiplications is omitted by improving the speed and power/efficiency characteristics of multipliers as it plays a key role. In this, the approach is to round the operands to nearest exponent of two. By this approximations are made for improving the speed and efficiency. Since the final outputs are used in two Image processing applications, i.e., image sharpening and smoothing. This can be performed at different design abstraction levels i.e., circuit, logic and architecture levels using different techniques, here we use function approximation method (e.g., modifying the Boolean function of a circuit), a number of approximating arithmetic building blocks, such as adders, multipliers have been suggested. Finally, It has added advantage that it can be used as common multiplier design for both signed and un-signed operations and reduces logic size and facilitates with less power and delay. Here we are using Verilog HDL and Xilinx ISE14.8 software tools for simulation and synthesis purpose.***

*Index Terms*—***Accuracy, approximate computing, energy efficient, error analysis, high speed, multiplier.***

## I. Introduction

ENERGY minimization is one of the main design requirements in almost any electronic systems, especially the portable ones such as smart phones, tablets, and different gadgets [1]. It is highly desired to achieve this minimization with minimal performance (speed) penalty [1]. Digital signal processing (DSP) blocks are key components of these portable devices for realizing various multimedia applications. The computational core of these blocks is the arithmetic logic unit where multiplications have the greatest share among all arithmetic operations performed in these DSP systems [2]. Therefore, improving the speed and power/energy-efficiency characteristics of multipliers plays a key role in improving the efficiency of processors.

Many of the DSP cores implement image and video processing algorithms where final outputs are either images or videos prepared for human consumptions. This fact enables us to use approximations for improving the speed/energy efficiency. This originates from the limited perceptual abilities of human beings in observing an image or a video. In addition to the image and video processing applications, there are other areas where the exactness of the arithmetic operations is not critical to the functionality of the system (see [3], [4]). Being able to use the approximate computing provides the designer with the ability of making tradeoffs between the accuracy and the speed as well as power/energy consumption [2], [5].

Applying the approximation to the arithmetic units can be performed at different design abstraction levels including circuit, logic, and architecture levels, as well as algorithm and software layers [2]. The approximation may be performed using different techniques such as allowing some timing violations(e.g., voltage over-scaling or over-clocking) and function approximation methods (e.g., modifying the Boolean function of a circuit) or a combination of them [4], [5]. In the category of function approximation methods, a number of approximating arithmetic building blocks, such as adders and multipliers, at different design levels have been suggested (see [6]–[8]).

In this paper, we focus on proposing a high-speed low-power/energy yet approximate multiplier appropriate for error

Thisp paper submitted on 22 december of 2020 for review by Manchiryala Manogna, M.TECH, Dept. of ECE(manchiryalamanogna369@gmail.com). This work was submitted in the conference of "The International Conference on Trends in Information, Management, Engineering and Sciences(ICTIMES), organized by the World Academic - Industry Research Collaboration Organization which takes place from 22nd January to the 23rd January 2020 at the Malla Reddy Engineering College in Hyderabad, India. The conference will cover areas like Computer Science and Engineering, Information Technology, Electronics and Communication Engineering, Mechanical Engineering and Aerospace and many more.

resilient DSP applications. The proposed approximate multiplier, which is also area efficient, is constructed by modifying the conventional multiplication approach at the algorithm level assuming rounded input values. We call this rounding-based approximate (RoBA) multiplier. The proposed multiplication approach is applicable to both signed and unsigned multiplications for which three optimized architectures are presented. The efficiencies of these structures are assessed by comparing the delays, power and energy consumptions, energy-delay products (EDPs), and areas with those of some approximate and accurate (exact) multipliers.
The contributions of this paper can be summarized as follows:

1) presenting a new scheme for RoBA multiplication by modifying the conventional multiplication approach;
2) describing three hardware architectures of the proposed approximate multiplication scheme for sign and unsigned operations.

The rest of this paper is organized as follows. Section II discusses the related works about approximate multipliers. The proposed scheme of the approximate multiplication, its hardware implementations, and its accuracy results are presented in Section III. In Section IV, the characteristics of the proposed approximate multiplier compared with the accurate and approximate multipliers, and also its effectiveness in image processing applications are studied. Finally, the conclusion is drawn in Section V.

## II. Prior Works

In this section, some of the previous works in the field of approximate multipliers are briefly reviewed. In [3], an approximate multiplier and an approximate adder based on a technique named broken-array multiplier (BAM) were proposed. By applying the BAM approximation method of [3] to the conventional modified Booth multiplier, an approximate signed Booth multiplier was presented in [5]. The approximate multiplier provided power consumption savings from 28% to 58.6% and area reductions from 19.7% to 41.8% for different word lengths in comparison with a regular Booth multiplier. Kulkarni *et al.* [6] suggested an approximate multiplier consisting of a number of $2 \times 2$ inaccurate building blocks that saved the power by 31.8%–45.4% over an accurate multiplier.

An approximate signed 32-bit multiplier for speculation purposes in pipelined processors was designed in [7]. It was 20% faster than a full-adder-based tree multiplier while having a probability of error of around 14%. In [8], an error-tolerant multiplier, which computed the approximate result by dividing the multiplication into one accurate and one approximate part, was introduced, in which the accuracies for different bit widths were reported. In the case of a 12-bit multiplier, a power saving of more than 50% was reported. In [9], two approximate 4:2 compressors for utilizing in a regular Dadda multiplier were designed and analyzed.

The use of approximate multipliers in image processing applications, which leads to reductions in power consumption, delay, and transistor count compared with those of an exact multiplier design, has been discussed in the literature. In [10], an accuracy-configurable multiplier architecture (ACMA) was suggested for error-resilient systems. To increase its throughput, the ACMA made use of a technique called carry-in prediction that worked based on a pre-computation logic.
When compared with the exact one, the proposed approximate multiplication resulted in nearly 50% reduction in the latency by reducing the critical path. Also, Bhardwaj *et al.* [11]presented an approximate Wallace tree multiplier (AWTM). Again, it invoked the carry-in prediction to reduce the critical path. In this work, AWTM was used in a real-time benchmark image application showing about 40% and 30% reductions in the power and area, respectively, without any image quality loss compared with the case of using an accurate Wallace tree multiplier (WTM) structure.

In [12], approximate unsigned multiplication and division based on an approximate logarithm of the operands have been proposed. In the proposed multiplication, the summation of the approximate logarithms determines the result of the operation. Hence, the multiplication is simplified to some shift and add operations. In [13], a method for increasing the accuracy of the multiplication approach of [12] was proposed.
It was based on the decomposition of the input operands. This method considerably improved the average error at the price of increasing the hardware of the approximate multiplier by about two times.

In [16], a dynamic segment method (DSM) is presented, which performs the multiplication operation on an *m*-bit segment starting from the leading one bit of the input operands.

A dynamic range unbiased multiplier (DRUM) multiplier, which selects an *m*-bit segment starting from the leading one bit of the input operands and sets the least significant bit of the truncated values to one, has been proposed in [17]. In this structure, the truncated values are multiplied and shifted to left to generate the final output. In [18], an approximate 4×4 WTM has been proposed that uses an inaccurate 4:2 counter. In addition, an error correction unit for correcting the outputs has been suggested. To construct larger multipliers, this 4×4 inaccurate Wallace multiplier can be used in an array structure.

Most of the previously proposed approximate multipliers are based on either modifying the structure or complexity reduction of a specific accurate multiplier. In this paper, similar to [12], we propose performing the approximate multiplication through simplifying the operation. The difference between our and [12] is that, although the principles in both works are almost similar for unsigned numbers, the mean error of our proposed approach is smaller. In addition, we suggest some approximation techniques when the multiplication is performed for signed numbers.

## III. Proposed Approximate Multiplier

### *Multiplication Algorithm of RoBA Multiplier*

The main idea behind the proposed approximate multiplier is to make use of the ease of operation when the numbers are two to the power n (2n). To elaborate on the operation of the approximate multiplier, first, let us denote the rounded numbers of the input of A and B by Ar and Br, respectively. The multiplication of A by B may be rewritten as

$$A \times B = (Ar - A) \times (Br - B) + Ar \times B + Br \times A - Ar \times Br. \quad (1)$$

The key observation is that the multiplications of Ar × Br, Ar

×B, and Br ×A may be implemented just by the shift operation. The hardware implementation of (Ar − A) × (Br − B),however, is rather complex. The weight of this term in the final result, which depends on differences of the exact numbers from their rounded ones, is typically small. Hence, we propose to omit this part from(1), helping simplify the multiplication operation. Hence, to perform the multiplication process, the following expression is used:

$$A \times B = Ar \times B + Br \times A - Ar \times Br. \qquad (2)$$

Thus, one can perform the multiplication operation using three shift and two addition/subtraction operations. In this approach, the nearest values for A and B in the form of 2n should be determined. When the value of A (or B) is equal to the 3 × 2p−2 (where p is an arbitrary positive integer larger than one), it has two nearest values in the form of 2n with equal absolute differences that are 2p and 2p−1. While both values lead to the same effect on the accuracy of the proposed multiplier, selecting the larger one (except for the case of p = 2) leads to a smaller hardware implementation for determining the nearest rounded value, and hence, it is considered in this paper. It originates from the fact that the numbers in the form of 3 × 2p−2 are considered as do not care in both rounding up and down simplifying the process, and smaller logic expressions may be achieved if they are used in the rounding up.

The only exception is for three, which in this case, two is considered as its nearest value in the proposed approximate multiplier. It should be noted that contrary to the previous work where the approximate result is smaller than the exact result, the final result calculated by the ROBA multiplier may be either larger or smaller than the exact result depending on the magnitudes of Ar and Br compared with those of A and B, respectively. Note that if one of the operands (say A) is smaller than its corresponding rounded larger than the exact result. This is due to the fact that, in this case, the multiplication result of (Ar − A) × (Br − B) will be negative. Since the difference between (1) and (2) is precisely this product, the approximate result becomes larger than the exact one. Similarly, if both A and B are larger or both are smaller than Ar and Br, then approximate result will be smaller than the exact result.

Finally, it should be noted the advantage of the proposed RoBA multiplier exists only for positive inputs because in the two's complement representation, the rounded values of negative inputs are not in the form of 2n. Hence, we suggest that, before the multiplication operation starts, the absolute values of both inputs and the output sign of the multiplication result based on the inputs signs be determined and then the operation be performed for unsigned numbers and, at the last stage, the proper sign be applied to the unsigned result. The hardware implementation of the proposed approximate multiplier is explained next.

## IV. HARDWARE IMPLEMENTATION OF RoBA MULTIPLIER

### *A. Signed RoBA Multiplier*

Based on (2), we provide the block diagram for the hardware implementation of the proposed multiplier in Fig. 1 where the inputs are represented in two's complement format. First, the signs of the inputs are determined, and for each negative value, the absolute value is generated. Next, the rounding block extracts the nearest value for each absolute value in the formof 2*n*. It should be noted that the bit width of the output of this block is *n* (the most significant bit of the absolute value of an *n*-bit number in the two's complement format is zero).

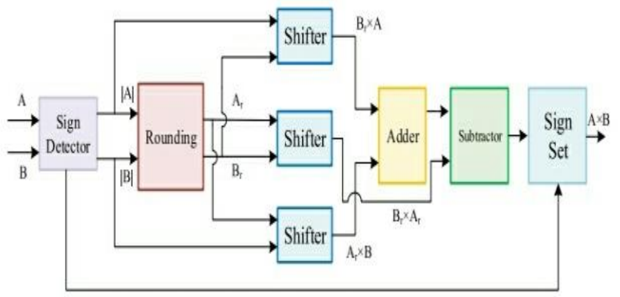


fig.Block diagram for the hardware implementation of the proposed signed multiplier.

To find the nearest value of input *A*, we use the following equation to determine each output bit of the rounding block:

$$Ar\,[n-1] = A[n-1] \cdot A[n-2] \cdot A[n-3] + A[n-1] \cdot A[n-2]$$

$$Ar\,[n-2] = (A[n-2] \cdot A[n-3] \cdot A[n-4] + A[n-2] \cdot A[n-3]) \cdot \overline{A[n-1]}$$

.
.
.

$$Ar\,[i] = (A[i] \cdot A[i-1] \cdot A[i-2] + A[i] \cdot A[i-1]) \cdot \prod_{i=i+1}^{n-1} \overline{A[i]}$$

.
.
.

$$Ar\,[3] = (A[3] \cdot A[2] \cdot A[1] + A[3] \cdot A[2]) \cdot \prod_{i=4}^{n-1} \overline{A[i]}$$

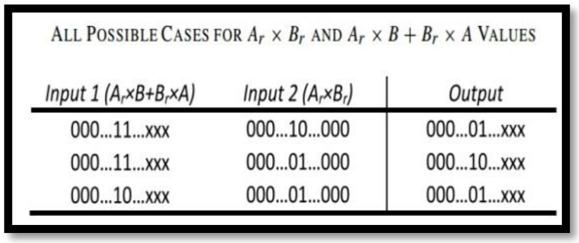
$$Ar\,[2] = A[2] \cdot A[1] \cdot \prod_{i=3}^{n-1} \overline{A[i]}$$

$$Ar\,[1] = A[1] \cdot \prod_{i=2}^{n-1} \overline{A[i]}$$

$$Ar\,[0] = A[0] \cdot \prod_{i=1}^{n-1} \overline{A[i]} \qquad (3)$$

In the proposed equation, *Ar* [*i* ] is one in two cases. In the first case, *A*[*i* ] is one and all the bits on its left side are zero while *A*[*i* − 1] is zero. In the second case, when *A*[*i*] and all its left-side bits are zero, *A*[*i* − 1] and *A*[*i* − 2] are both one. Having determined the rounding values, using three barrel shifter blocks, the products *Ar* × *Br* , *Ar* × *B*, and *Br* × *A* are calculated. Hence, the amount of shifting is determined based on $\log_2 A_r - 1$ (or $\log_2 B_r - 1$) in the case of *A* (or *B*) operand. Here, the input bit width of the shifter blocks is *n*, while their outputs are 2*n*.

A single 2*n*-bit Kogge-Stone adder is used to calculate the summation of *Ar* × *B* and *Br* × *A*. The output of this adder and the result of *Ar* × *Br* are the inputs of the *subtractor* block whose output is the absolute value of the output of the proposed multiplier. Because *Ar* and *Br* are in the form of 2*n*,the inputs of the *subtractor* may take one of the three input patterns shown in Table I. The corresponding output pattern



ALL POSSIBLE CASES FOR $A_r \times B_r$ AND $A_r \times B + B_r \times A$ VALUES

| Input 1 ($A_r$×B+$B_r$×A) | Input 2 ($A_r$×$B_r$) | Output |
|---|---|---|
| 000...11...xxx | 000...10...000 | 000...01...xxx |
| 000...11...xxx | 000...01...000 | 000...10...xxx |
| 000...10...xxx | 000...01...000 | 000...01...xxx |

are also shown in Table I.

The forms of the inputs and output inspired us to conceive a simple circuit based on the following expression:

out = *(P* XOR *Z)* AND ({*(P* _ 1*)* XOR *(P* XOR *Z)*} or {*(P* AND *Z)* _ 1}*)* (4)

where P is Ar × B + Br × A and Z is Ar × Br. The corresponding circuit for implementing this expression is smaller and faster than the conventional subtraction circuit.

Finally, if the sign of the final multiplication result should be negative, the output of the subtractor will be negated in the sign set block. To negate values, which have the two's complement representation, the corresponding circuit based on $\bar{X}$ +1 should be used. To increase the speed of negation operation one may skip the incrementation process in the negating phase accepting it's associated error. As will be seen later, the significance of the error decreases as the input widths increases. In this paper, if the negation is performed exactly (approximately), the implementation is called signed RoBA (S-RoBA)multiplier [approximate S-RoBA (AS-RoBA) multiplier].

### *B. Un-Signed RoBA Multiplier*

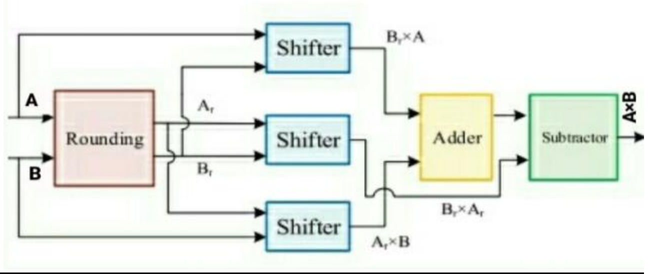


fig. Block diagram for the hardware implementation of the proposed un-signed multiplier

we provide the block diagram for the hardware implementation of the proposed multiplier in Fig. where the inputs are given to rounding block. Next, the rounding block extracts the nearest value for each absolute value in the form of 2^n. It should be noted that the bit width of the output of this block is n (the most significant bit of the absolute value of an n-bit number in the two's complement format is zero).

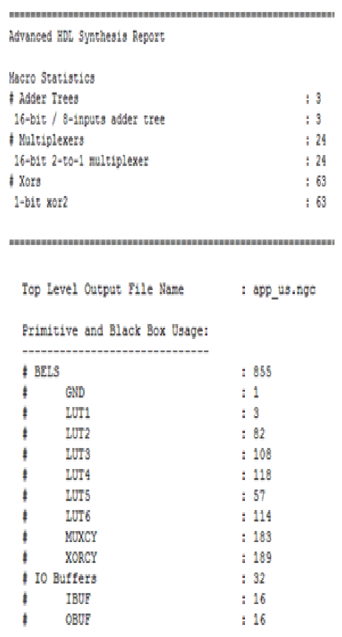
Having determined the rounding values, using three barrel shifter blocks, the products Ar × Br, Ar × B, and Br × A are calculated. Hence, the amount of shifting is determined based on logAr2 − 1 (or logBr2 − 1) in the case of A (or B) operand. Here, the input bit width of the shifter blocks is n, while their outputs are 2n.

A single 2n-bit Brent-kung adder is used to calculate the summation of Ar × B and Br × A. The output of this adder and the result of Ar × Br are the inputs of the subtractor block whose output is the absolute value of the output of the proposed multiplier. Because Ar and Br are in the form of 2n,the inputs of the subtractor may take one of the three input patterns shown in Table 3.1.

In the case where the inputs are always positive, to increase the speed and reduce the power consumption, the sign detector and sign set blocks are omitted from the architecture, providing us with the architecture called unsigned RoBA (U-RoBA) multiplier. In this case, the output width of the rounding block is n + 1 where this bit is determined based on Ar[n] = A[n − 1] · A[n − 2]. This is because in the case of unsigned 11x ... x (where x denotes do not care) with the bit width of n, its rounding value is 10…0 with the bit width of n + 1. Therefore, the input bit width of the shifters is n + 1. However, because the maximum amount of shifting is n − 1, 2n is considered for the output bit width of the shifters.

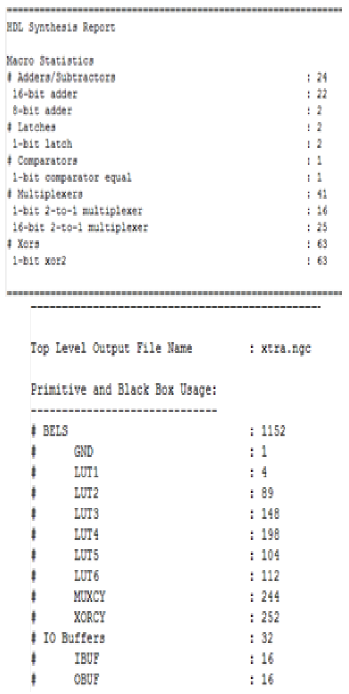
## V. RESULTS AND DISCUSSION

### *A. HDL Synthesis Report for Existing and Proposed RoBA Multipliers:*



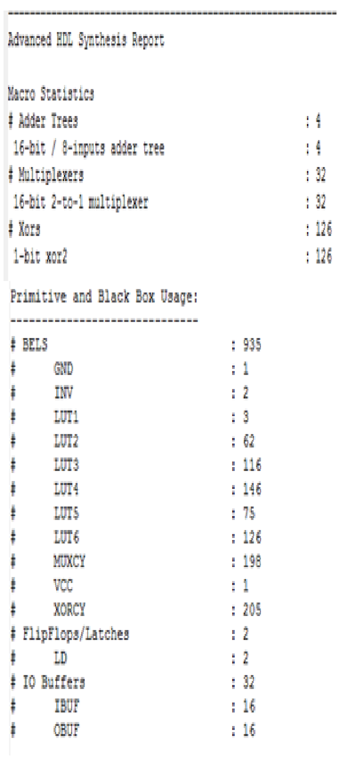
```
=====================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adder Trees                                              : 3
 16-bit / 8-inputs adder tree                              : 3
# Multiplexers                                             : 24
 16-bit 2-to-1 multiplexer                                 : 24
# Xors                                                     : 63
 1-bit xor2                                                : 63

=====================================================================

  Top Level Output File Name            : app_us.ngc

  Primitive and Black Box Usage:
  ------------------------------
  # BELS                                : 855
  #      GND                            : 1
  #      LUT1                           : 3
  #      LUT2                           : 82
  #      LUT3                           : 108
  #      LUT4                           : 118
  #      LUT5                           : 57
  #      LUT6                           : 114
  #      MUXCY                          : 183
  #      XORCY                          : 189
  # IO Buffers                          : 32
  #      IBUF                           : 16
  #      OBUF                           : 16
```

Table A: HDL Synthesis Report for approximate un-signed RoBA Multiplier



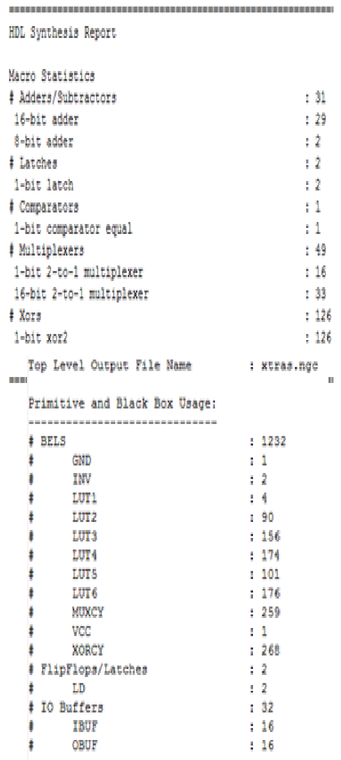
```
=====================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                       : 24
 16-bit adder                                              : 22
 8-bit adder                                               : 2
# Latches                                                  : 2
 1-bit latch                                               : 2
# Comparators                                              : 1
 1-bit comparator equal                                    : 1
# Multiplexers                                             : 41
 1-bit 2-to-1 multiplexer                                  : 16
 16-bit 2-to-1 multiplexer                                 : 25
# Xors                                                     : 63
 1-bit xor2                                                : 63

=====================================================================
   --------------------------------------------------------

   Top Level Output File Name            : xtra.ngc

   Primitive and Black Box Usage:
   ------------------------------
   # BELS                                : 1152
   #      GND                            : 1
   #      LUT1                           : 4
   #      LUT2                           : 89
   #      LUT3                           : 148
   #      LUT4                           : 198
   #      LUT5                           : 104
   #      LUT6                           : 112
   #      MUXCY                          : 244
   #      XORCY                          : 252
   # IO Buffers                          : 32
   #      IBUF                           : 16
   #      OBUF                           : 16
```

Table B : HDL Synthesis Report for Accurate un-signed RoBA Multiplier



```
--------------------------------------------------------------
Advanced HDL Synthesis Report

Macro Statistics
# Adder Trees                                       : 4
 16-bit / 8-inputs adder tree                       : 4
# Multiplexers                                      : 32
 16-bit 2-to-1 multiplexer                          : 32
# Xors                                              : 126
 1-bit xor2                                         : 126

Primitive and Black Box Usage:
------------------------------
# BELS                              : 935
#      GND                          : 1
#      INV                          : 2
#      LUT1                         : 3
#      LUT2                         : 62
#      LUT3                         : 116
#      LUT4                         : 146
#      LUT5                         : 75
#      LUT6                         : 126
#      MUXCY                        : 198
#      VCC                          : 1
#      XORCY                        : 205
# FlipFlops/Latches                 : 2
#      LD                           : 2
# IO Buffers                        : 32
#      IBUF                         : 16
#      OBUF                         : 16
```

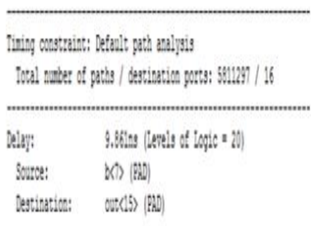
Table C: HDL Synthesis Report for approximate signed RoBA Multiplier



```
==============================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                : 31
 16-bit adder                                       : 29
 8-bit adder                                        : 2
# Latches                                           : 2
 1-bit latch                                        : 2
# Comparators                                       : 1
 1-bit comparator equal                             : 1
# Multiplexers                                      : 49
 1-bit 2-to-1 multiplexer                           : 16
 16-bit 2-to-1 multiplexer                          : 33
# Xors                                              : 126
 1-bit xor2                                         : 126

   Top Level Output File Name           : xtras.ngc

   Primitive and Black Box Usage:
   ------------------------------
   # BELS                              : 1232
   #      GND                          : 1
   #      INV                          : 2
   #      LUT1                         : 4
   #      LUT2                         : 90
   #      LUT3                         : 156
   #      LUT4                         : 174
   #      LUT5                         : 101
   #      LUT6                         : 176
   #      MUXCY                        : 259
   #      VCC                          : 1
   #      XORCY                        : 268
   # FlipFlops/Latches                 : 2
   #      LD                           : 2
   # IO Buffers                        : 32
   #      IBUF                         : 16
   #      OBUF                         : 16
```

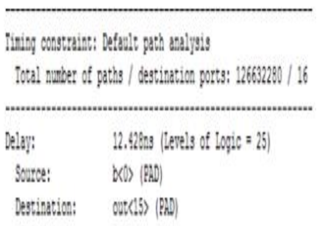
Table D : HDL Synthesis Report for Accurate signed RoBA Multiplier

*B. Timing Delays for Existing and Proposed RoBA Multipliers:*



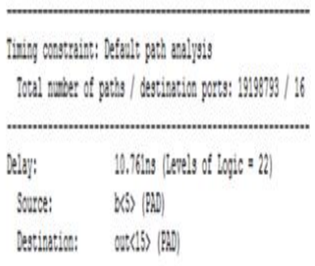
```
--------------------------------------------------------------
Timing constraint: Default path analysis
  Total number of paths / destination ports: 5811297 / 16
--------------------------------------------------------------
Delay:               9.861ns (Levels of Logic = 20)
  Source:            b<7> (PAD)
  Destination:       out<15> (PAD)
```

Table E: Timing delay for approximate un-signed RoBA Multiplier



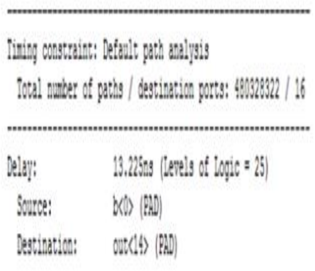
```
--------------------------------------------------------------
Timing constraint: Default path analysis
  Total number of paths / destination ports: 126632280 / 16
--------------------------------------------------------------
Delay:               12.428ns (Levels of Logic = 25)
  Source:            b<0> (PAD)
  Destination:       out<15> (PAD)
```

Table F: Timing delay for approximate signed RoBA Multiplier



```
--------------------------------------------------------------
Timing constraint: Default path analysis
  Total number of paths / destination ports: 19198793 / 16
--------------------------------------------------------------
Delay:               10.761ns (Levels of Logic = 22)
  Source:            b<5> (PAD)
  Destination:       out<15> (PAD)
```

Table G: Timing delay for accurate un-signed RoBA Multiplier



```
--------------------------------------------------------------
Timing constraint: Default path analysis
  Total number of paths / destination ports: 480328322 / 16
--------------------------------------------------------------
Delay:               13.225ns (Levels of Logic = 25)
  Source:            b<0> (PAD)
  Destination:       out<14> (PAD)
```

Table H: Timing delay for accurate signed RoBA Multiplier

*C. Comparision Table:*

| S . No | TYPES | NAME OF THE MULTIPLI-ER | No. OF XOR'S | NO.OF LUT'S | NO.OF IO BUF-FERS | DELAY | MEMORY OCCUPI-ED |
|---|---|---|---|---|---|---|---|
| 1 | EXACT RoBA MULTIPLI-ER | SIGNED RoBA ACCURA-TE | 126 | 701 | 32 | 13.225 ns | 414820 kb |
|  |  | UN-SIGNED RoBA ACCURATE | 126 | 655 | 32 | 10.761 ns | 414628 kb |
| 2 | APPROXIMATE RoBA MULTIPLIE-R | SIGNED RoBA APPROXI-MATE | 63 | 528 | 32 | 12.428 ns | 414052 kb |
|  |  | UN-SIGNED RoBA APPROXI-MATE | 63 | 482 | 32 | 9.861ns | 413924 kb |

From the above reports, We can conclude that in the proposed Approximate Un-Signed and Signed RoBA Multiplier the number of gates and the delay has been reduced compared to accurate multiplier.

Hence,Design complexity is less. It is applicable to both signed and unsigned multiplications. The mean error is smaller, compared to other multipliers. Area has been reduced. The number of gates has been reduced compared to accurate multiplier.

## V1.Conclusion

We proposed a high-speed yet energy efficient approximate multiplier called RoBA multiplier. The proposed multiplier, which had high accuracy, was based on rounding of the inputs in the form of 2^n. In this way, the computational intensive part of the multiplication was omitted improving speed and energy consumption at the price of a small error. The proposed approach is applicable to both signed and unsigned multiplications. The efficiencies of the proposed multipliers were evaluated by comparing them with those of approximate RoBA and accurate RoBA multipliers using different design parameters. The results revealed that,in most (all) cases, the RoBA multiplier architectures outperformed the corresponding approximate (exact) multipliers. The comparison revealed almost same image qualities as those of exact multiplication algorithms.

## References

[1] M. Alioto, "Ultra-low power VLSI circuit design demystified and explained: A tutorial," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59,no. 1, pp. 3–29, Jan. 2012.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137,Jan. 2013.

[3] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient

VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*,
vol. 57, no. 4, pp. 850–862, Apr. 2010.
[4] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2011, pp. 667–673.
[5] F. Farshchi, M. S. Abrishami, and S. M. Fakhraie, "New approximate multiplier for low power digital signal processing," in *Proc. 17th Int. Symp. Comput. Archit. Digit. Syst. (CADS)*, Oct. 2013, pp. 25–30.
[6] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an under designed multiplier architecture," in *Proc. 24th Int. Conf. VLSI Design*, Jan. 2011, pp. 346–351.
[7] D. R. Kelly, B. J. Phillips, and S. Al-Sarawi, "Approximate signed binary integer multipliers for arithmetic data value speculation," in *Proc. Conf. Design Archit. Signal Image Process.*, 2009, pp. 97–104.
[8] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. IEEE Int. Conf. Electron evices Solid-State Circuits (EDSSC)*, Dec. 2010, pp. 1–4.
[9] A. Momeni, J. Han, P. Montuschi, and F. Lombardi,
"Design and analysis of approximate compressors for
multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, Apr. 2015.
[10] K. Bhardwaj and P. S. Mane, "ACMA: Accuracy-configurable multiplier
architecture for error-resilient system-on-chip," in *Proc. 8th Int.*
*Workshop Reconfigurable Commun.-Centric Syst.-Chip*, 2013, pp. 1–6.
[11] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and
area-efficient approximate wallace tree multiplier for error-
resilient systems," in *Proc. 15th Int. Symp. Quality Electron. Design (ISQED)*, 2014, pp. 263–269.
[12] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 4, pp. 512–517, Aug. 1962.
[13] V. Mahalingam and N. Ranganathan, "Improving accuracy in Mitchell's ogarithmic multiplication using operand decomposition," *IEEE Trans. Comput.*, vol. 55, no. 12, pp. 1523–1535, Dec. 2006.
[14] *Nangate 45nm Open Cell Library*, accessed on 2010. [Online]. Available: http://www.nangate.com/
[15] H. R. Myler and A. R. Weeks, *The Pocket Handbook of Image Processing Algorithms in C*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2009.
[16] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim,
"Energy-efficient approximate multiplication for digital signal processing
and classification applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1180–1184, Jun. 2015.
[17] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range
unbiased multiplier for approximate applications," in *Proc. IEEE/ACM*
*Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, 2015,
pp. 418–425.
[18] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with
error correction," in *Proc. 31st Int. Conf. Comput. Design (ICCD)*, 2013,

**MANCHIRYALA MANOGNA is currently pursuing Post Graduation in Malla Reddy college of Engineering(MRCE), Maisammaguda, Hyderabad and completed completed her Under Graduation student of Kakatiya Institute of Technology and Science for Women(KITSW) Implemented this paper as her project during her under graduation and continued works in the post graduation and submitted paper for the conference in the ICTIMES 2020 held at Malla Reddy college of Engineering**. .