Exploring Approximate Integer Multiplication

Vishal Vijay Devadiga, CS21BTECH11061
Guide: Dr. Rajesh Kedia

Abstract
Approximate Integer Multipliers are used in many applications such as digital signal processing, image processing, and machine learning. In this report, I will:

- Explore the current state of the art in Approximate Integer Multiplier design.
- Present two Approximate Integer Multiplier design based on ROBA (Rounding Based Approximate) Multiplier that improve upon the original design in some aspects.

Contents

1 Introduction

In some applications, the accuracy of the multiplier is not as important as the power consumption or area. In such cases, rather than using a full-precision multiplier, an approximate multiplier can be used. Approximate multipliers are used in many applications such as digital signal processing, image processing, and machine learning.
In this report, I will explore the current state of the art in Approximate Integer Multiplier design.

1.1 Logarithmic Based Approximate Multipliers

Logarithmic based approximate multipliers are a type of approximate multiplier that uses logarithmic features to reduce the number of partial products. There are many different types of logarithmic based approximate multipliers such as: DRUM, Wallace Tree, Baugh-Wooley Multiplier, DSM, Mitchell's Logarithmic Multiplier, ROBA Multiplier .etc.

All of these multipliers have their own advantages and disadvantages. Below is a brief comparison of some of the logarithmic based approximate multipliers.

Multiplier	**Signed**	**Max Error**	**MRE**	**Power**	**Delay**	**Area**
S-ROBA[4]	Yes	11.11%	2.91%	2.09	1.21	2.90
U-ROBA[4]	No	11.11%	2.92%	1.23	1.00	2.32
DRUM6[1]	No	6.34%	1.47%	1.20	1.17	1.00
DSM8[2]	No	9.99%	0.53%	1.00	1.29	1.53
HAAM[3]	No	13.76%	2.04%	4.01	2.63	4.07

Table 1: Comparison of some logarithmic based approximate multipliers

---

2 ROBA Multiplier

Rounding based approximate multiplier

2.1 What is ROBA?

ROBA (Rounding Based Approximate) Multiplier is a type of approximate multiplier that replaces multiplication with addition and shift operations. Consider the number X. For this number, X_r is defined to be the nearest power of 2 to X. For example, if:

- $X = 5$, then $X_r = 4$
- $X = 7$, then $X_r = 8$
- $X = 10$, then $X_r = 8$

A multiplication operation for A and B can be written as:

$$A \times B = (A - A_r) \times (B - B_r) + A_r \times B + A \times B_r - A_r \times B_r \tag{1}$$

Considering that A_r and B_r are powers of 2, the 3 terms in the equation can be calculated using only addition and shift operations. The original paper[4] states that on ignoring the term $(A - A_r) \times (B - B_r)$, the multiplication operation can be approximated with some shift and add operations.

$$A \times B \approx A_r \times B + A \times B_r - A_r \times B_r \tag{2}$$

Thus, in ROBA Multiplier, the following operations are performed:

- Check the sign of the product and calucate $A = |A|$, $B = |B|$
- Calculate A_r and B_r using the rounder module
- Calculate $A_r \times B$, $A \times B_r$ and $A_r \times B_r$ using the shifter module
- Calculate the final product using the adder module and subtractor module
- Check the sign of the product and output the final product

2.2 Design of ROBA

Below is the image from the original paper[4] that shows the architecture of the ROBA Multiplier.

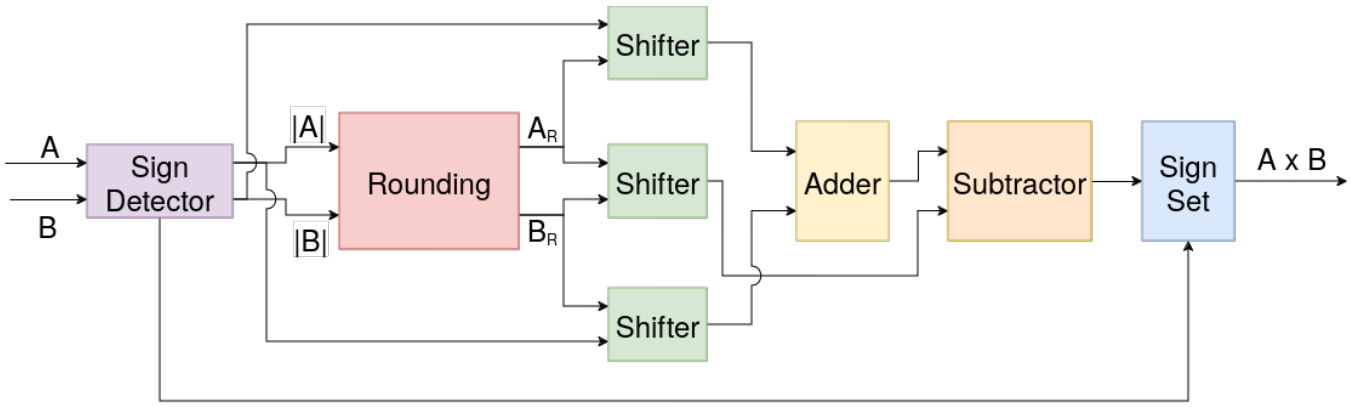

Figure 1: ROBA Multiplier Architecture

2.2.1 Sign Detector Module

Checking for the sign of the product is done using the sign detector module. The sign detector module checks the sign of the input numbers by checking the last bit of the input numbers. If the last bit is 1, then the number is negative, else the number is positive.

It sets the sign bit of the output product to 1 if the product is negative, else it sets the sign bit to 0. It also sets the sign bit of the input numbers to 0.

2.2.2 Rounder Module

The rounder module calculates the nearest power of 2 to the input number. For a number X, the rounder module calculates X_r such that:

$$X_r[n-1] = \overline{X[n-1]} \bullet X[n-2] \bullet X[n-3] + X[n-1] \bullet \overline{X[n-2]} \tag{3}$$

$$X_r[n-2] = (\overline{X[n-2]} \bullet X[n-3] \bullet X[n-4] + X[n-2] \bullet \overline{X[n-3]}) \bullet \overline{X[n-1]} \tag{4}$$

$$X_r[i] = (\overline{X[i]} \bullet X[i-1] \bullet X[i-2] + X[i] \bullet \overline{X[i-1]}) \bullet \Pi_{j=i+1}^{n} \overline{X[j]} \tag{5}$$

$$X_r[2] = (X[2] \bullet \overline{X[1]}) \bullet \Pi_{j=3}^{n} \overline{X[j]} \tag{6}$$

$$X_r[1] = X[1] \bullet \Pi_{j=2}^{n} \overline{X[j]} \tag{7}$$

$$X_r[0] = X[0] \bullet \Pi_{j=1}^{n} \overline{X[j]} \tag{8}$$

Basically the motive is to find the first 1 (MSB) from the left and set all the bits to the right of it to 0. This is done by the terms in the equations above.

The Pi term is the product of all the bits to the left of the current bit. This ensures that all the bits to the right of the first 1 are set to 0. The other terms in the equations are used to find the pattern of the bits that is, either 011 or 10.

This module is used to calculate A_r and B_r.

2.2.3 Subtractor Module

The subtractor module calculates the term $(A_r \times B + A \times B_r) - A_r \times B_r$. The term $(A_r \times B + A \times B_r)$ is calculated using the adder module.

The subtractor module is not a subtractor in the traditional sense. Considering that there are constraints on A, B, A_r, B_r, that is, there are definite patterns in the bits of A, B, A_r, B_r, the subtractor module can be implemented using binary operations.

Input 1 $(A_r \times B + A \times B_r)$	**Input 2** $(A_r \times B_r)$	**Output**
$000\ldots011x\ldots xxx$	$000\ldots010x\ldots xxx$	$000\ldots001x\ldots xxx$
$000\ldots011x\ldots xxx$	$000\ldots001x\ldots xxx$	$000\ldots010x\ldots xxx$
$000\ldots010x\ldots xxx$	$000\ldots001x\ldots xxx$	$000\ldots001x\ldots xxx$

Table 2: Cases for the subtractor module

Considering the cases in the table above, the subtractor module can be implemented using binary operations. Let:

- $P = A_r \times B + A \times B_r$
- $Q = A_r \times B_r$

Then the output of the subtractor module is:

$$output = (P \oplus Q) \wedge ([(P \ll 1) \oplus (P \oplus Q)] \vee [P \wedge Q \ll 1]) \tag{9}$$

2.3 Evaluation of ROBA

ROBA Multiplier ignores the term $(A - A_r) \times (B - B_r)$ in the multiplication operation. This term is the error term in the multiplication operation. The error term is the difference between the actual product and the approximate product. The error term is calculated as:

$$Error = Actual - ApproximateActual \times 100 \tag{10}$$

Thus,

$$MRE = (A - A_r) \times (B - B_r)A \times B \times 100 \tag{11}$$

The worst case error for the ROBA Multiplier is 11.11% when the terms A and B are of form 3×2^n.

---

3 Extended ROBA Multiplier

Approximate the $(A - A_r) \times (B - B_r)$ term

Suppose, rather than ignoring the term $(A-A_r)\times(B-B_r)$, we approximate it using some operations. This leads to the Extended ROBA Multiplier. Let:

- $A' = A - A_r$
- $B' = B - B_r$

Then, the multiplication operation can be written as:

$$A \times B = A' \times B' + A_r \times B + A \times B_r - A_r \times B_r \tag{12}$$

The term $A' \times B'$ can be approximated using ROBA Multiplier once again.

Below are 2 architectures for the Extended ROBA Multiplier:

3.1 TD-ROBA

The main motivation behind the TD-ROBA Multiplier is to parallelize the calculation of the term $A' \times B'$ in the Extended ROBA Multiplier. This is done by using 2 ROBA Multipliers in parallel. Below is the architecture of the TD-ROBA Multiplier:

In order to reduce the delay of the TD-ROBA Multiplier, the Rounder Module is changed to round off numbers to the nearest floor power of 2. That is,

- $X = 5$, then $X_r = 4$
- $X = 7$, then $X_r = 4$
- $X = 10$, then $X_r = 8$

This is done so that, the term $A - A_r$ is calculated faster, as now,

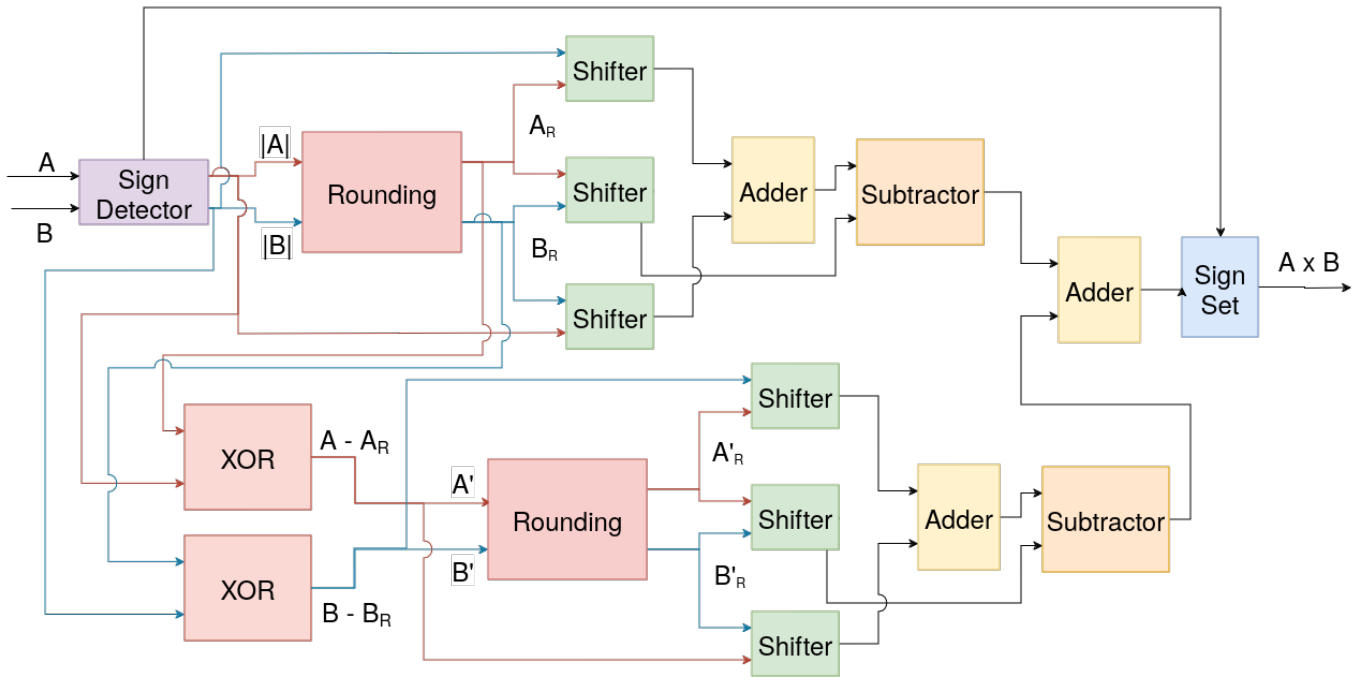

Figure 2: TD-ROBA Multiplier Architecture

- $A' = A - A_r = A \oplus A_r$
- $B' = B - B_r = B \oplus B_r$

Using the XOR operation, the term $A' \times B'$ can be calculated in parallel.

3.1.1 Rounder Module

The rounder module calculates the nearest floor power of 2 to the input number. The design for this module now simplifies to a priority encoder, which encodes the highest bit of the input number to 1 and the rest of the bits to 0.

This also simplifies calculation of the amount of bit shifts required to calculate the term $A_r \times B$, $A \times B_r$ and $A_r \times B_r$.

3.1.2 Subtractor Module

Considering that now, the constraints on the terms is even more strict, the subtractor module is also simplified. Let:

- $P = A_r \times B + A \times B_r$
- $Q = A_r \times B_r$

Then, the final product can be calculated as:

$$output = (P \oplus Q) \wedge \neg((\neg P \wedge Q) \ll 1) \tag{13}$$

3.1.3 Final Product

On calculating the individual ROBA terms in parallel, the final product is just the addition of the 2 terms. The final product is then checked for the sign and output accordingly.

3.1.4 Evaluation of TD-ROBA

Considering that the TD-ROBA Multiplier only rounds down to the nearest power of 2, the max error is expected to be higher than the ROBA Multiplier. However, this is compensated by the fact that the term $A' \times B'$ is also calculated using the multiplier rather than just ignoring it. This leads to a reduction in the average error and delay of the TD-ROBA Multiplier.

The max error of the TD-ROBA Multiplier is 6.25%. This is lower than the ROBA Multiplier, but higher than the R-ROBA Multiplier.

3.2 R-ROBA

The primary motivation behind the R-ROBA Mutliplier is to minimize the area overhead of the Extended ROBA Mutliplier while simultaneously reducing its maximum error. This is achieved by reusing the **ROBA Multiplier** for computing the additional term $A' \times B'$. The architectural design of the R-ROBA Multiplier is illustrated below:

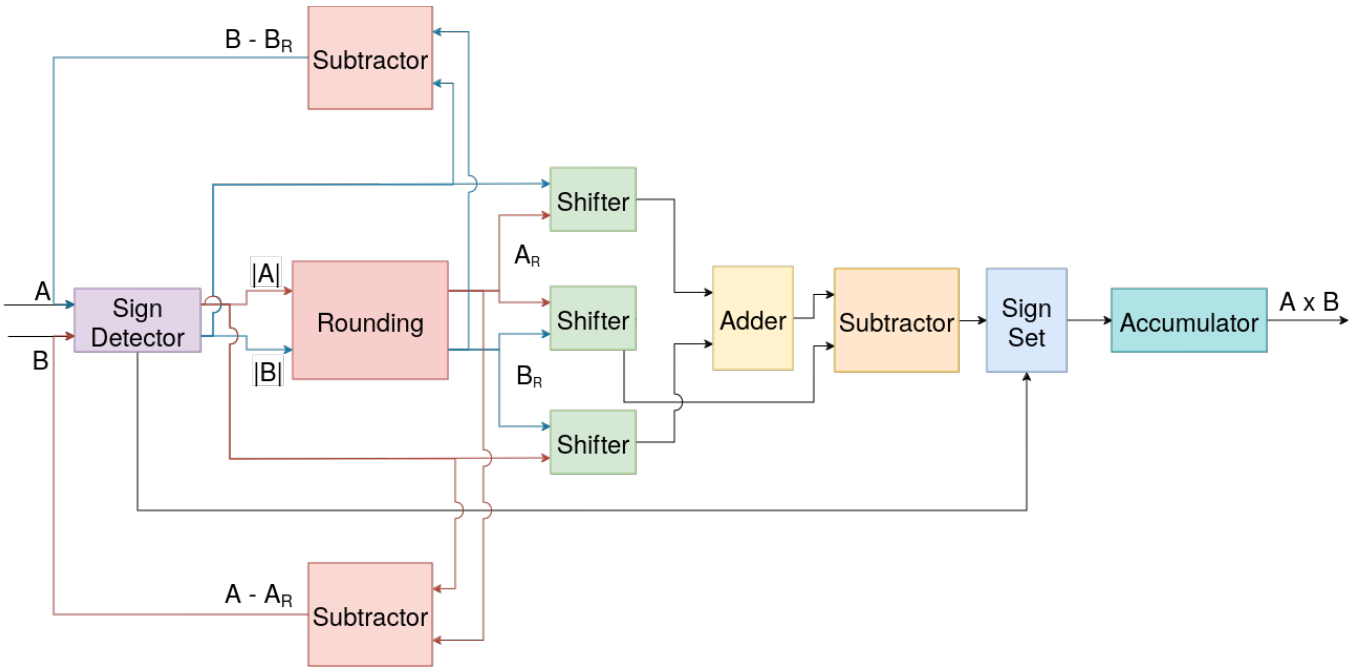

Figure 3: R-ROBA Multiplier Architecture

The reuse of the existing ROBA Multiplier for the computation of $A' \times B'$ introduces a significant advantage by lowering the maximum error of the Extended ROBA Mutliplier. However, this approach also introduces a trade-off in terms of increased computational delay. Specifically, the delay is

expected to increase by approximately a factor of two, as the term $A' \times B'$ is computed sequentially rather than in parallel.

Despite this trade-off, the R-ROBA Multiplier presents a promising solution for applications requiring a balance between accuracy and hardware efficiency. Its design underscores the importance of resource sharing in reducing area while improving performance metrics such as error tolerance. Future optimization strategies could focus on mitigating the delay increase to further enhance the feasibility of this approach in real-world applications.

Such a multiplier reduces the max error rate to 1.21%, which is much lower compared to the original ROBA multiplier.

---

4 Evaluation

How does the Extended ROBA Multiplier compare to the ROBA Multiplier?

4.1 Evaluation Metrics

For evaluating such multipliers, performance metrics like Area, Power, Delay, Error, Mean Relative Error (MRE) are considered. These metrics help in understanding the trade-offs between the different multipliers.

4.2 Tools Used

Two suites were used for the evaluation: Open Source EDA tools and Cadence Suite of Tools. The Open Source EDA tools used were: Yosys, ABC, OpenROAD, SiliconCompiler. These tools were used for synthesis and place and route of the Verilog code.

4.3 Results

For this evaluation, TD-ROBA multiplier was implemented using Verilog and synthesized. Unfortunately, due to lack of time, R-ROBA multiplier could not be implemented and has been kept as a future work to do. The results are as follows:

Multiplier	**Signed**	**Max Error**	**MRE**	**Power**	**Delay**	**Area**
S-ROBA	Yes	11.11%	2.91%	1.70	1.21	1.25
U-ROBA	No	11.11%	2.92%	1.00	1.00	1.00
TD-ROBA	Yes	6.25%	0.96%	4.34	1.08	2.09

As per the table shown, the TD-ROBA design performs better in the max-error, mean-error and delay but with tradeoffs in area and power.

---

5 Conclusions

Based on the results, the DTROBA multiplier outperforms the RROBA multiplier in terms of maximum error, average error, and delay, albeit at the cost of higher area and power consumption. Additionally, the RROBA multiplier demonstrates better performance than the ROBA multiplier in terms of maximum error and average error.

5.1 Future Work

: In future, we can:

- Design and implement the RROBA multiplier to compare its performance with both ROBA and DTROBA multipliers, enabling a deeper analysis of trade-offs in error, delay, area, and power.
- Explore the design space of logarithmic-based approximate multipliers to identify potential improvements in performance and efficiency, especially for applications that can tolerate some computational inaccuracy.

By systematically evaluating and optimizing these designs, we can better understand the trade-offs between accuracy, speed, power, and area, paving the way for more efficient approximate computing architectures.

References

[1] Soheil Hashemi, R. Iris Bahar, and Sherief Reda. Drum: A dynamic range unbiased multiplier for approximate applications. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 418–425, 2015.

[2] Srinivasan Narayanamoorthy, Hadi Asghari Moghaddam, Zhenhong Liu, Taejoon Park, and Nam Sung Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(6):1180–1184, 2015.

[3] Zhixi Yang, Jun Yang, Xianbin Li, Jian Wang, and Zhou Zhang. A high accuracy approximate multiplier with error correction. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020.

[4] Reza Zendegani, Mehdi Kamal, Milad Bahadori, Ali Afzali-Kusha, and Massoud Pedram. Roba multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):393–401, 2017.