ISSN: 2278-4632
Vol-12 Issue-06 No.03 June 2022

SPEED EFFICIENT VLSI ARCHITECTURE OF TRUNCATION AND ROUNDING BASED SCALABLE APPROXIMATE MULTIPLIER

Kumbidi Tarpana Research Scholar, Sri Sivani College of Engineering, Chilakapalem, Andhra Pradesh 532402
Dr. V.Suryanarayana Associate professor, Sri Sivani College of Engineering, Chilakapalem, Andhra Pradesh 532402

ABSTRACT
Modern society is advancing in its pursuit of the minimum area challenge. Traditional designs use a large number of efficient structures in order to include the maximum speed. Because of their size, the great majority of constructions will have the same multiplier as the basic blocks, but will run slower. In real life, not all applications, like image processing and digital signal processing, need exact results. Approximative multipliers are utilised as a consequence. Based on these two things we introduce the Truncation-And-Round-Depending Scalable Approximate Multiplier (TOSAM), which decreases the number of partial products by rounding each input operand based on its leading one-bit location. Shifts, adds, and short fixed-width multiplication operations are used in the proposed architecture to execute multiplication. This is much faster than traditional multipliers. To increase overall accuracy, the multiplication part's input operands are rounded to the closest even value. Because input operands are reduced in accordance with their leading one-bit positions, operand width has only a little impact on accuracy, and the multiplier is scalable. The design parameter has undergone significant improvements (less area).
Index terms: carry look ahead adder, arithmetic unit, leading one detector, shift unit, approximate absolute unit, truncated unit and sign detector.

1. INTRODUCTION

A multiplication procedure is often divided into three phases. The input operands are used to construct partial products in the first phase. Imperfect goods increase in the second phase until only two rows remain. Using a (rapid) adder, the last two rows are totaled in the final step. The approximation may be applied to each of these phases. In the first phase, approximation may be used to cut down on the quantity of partial products or the complexity of their creation. In the second stage of multiplication Approximation can be used to lower the latency or power use of the reduction levels. One of these ways is to use an approximation compressor. The design of the adder used in the last step of multiplying has a big effect on how long the process takes and how much power it uses. As a result, in the last step, an approximation adder may be used to lower the multiplier's power consumption. In this paper, we give a way to figure out how many partial products there are. The suggested approximation method shortens input operands to h or t bits, depending on where their leading bit is, and these shortened values are used for multiplication and addition. To make the reduction levels less slow or use less power, approximation may be employed. Approximation compressors are one of these ways. The design of the adder used in the last step of multiplying has a big effect on how long the operation takes and how much power it needs. As a result, in the last step, an approximation adder may be used to lower the multiplier's power consumption. In this work, we show how to figure out how many partial products there are. The proposed approximation method cuts the input operands down to h or t bits, depending on where their leading bit is, and then uses these truncated values for multiplication and addition. Approximation can be used to make the reduction levels run faster or use less power. One of these ways is to use an approximation compressor. The design of the adder used in the last step of multiplying has a big effect on how long the operation takes and how much power it needs. As a result, in the last step, an approximation adder may be used to lower the multiplier's power consumption.

2. LITERATURE SURVEY

At multiple levels, exposing approximation computing advancements: From a behavioral to a gate-level perspective, authored by S. Xu and B. C. Schafer Many applications have a high tolerance for computational imprecision. Image processing, multimedia, and machine learning are a few examples. Circuits that consume less power, have a smaller footprint, and perform better might make advantage of these faults. Approximation optimizations in VLSI design were often restricted to a certain abstraction level or stage in previous studies. This research shows that a technique that uses more than one level is much better. As a consequence, different optimizations are used at each level in this investigation giving higher results when compared to single-level approaches. Furthermore, approximation computing is strongly dependent on data. We analyse the stability of approximation circuits in this study, The circuit is set up to handle a certain amount of data, but the actual load is not the same as what was expected. Researchers in the past have mostly focused on a single input data distribution because they thought it was a good representation of the whole workload. The findings suggest that our technique can find better and more optimum designs than earlier studies, as well as circuits that are more durable under changing workload scenarios.

3. PROPOSED APPROXIMATEMULTIPLIER TOSAM

Each integer positive (N) may be represented as

$$N = \sum_{i=0}^{k} 2^i x_i \qquad (1)$$

where k is the leading bit's location and xi is the N's i-th bit Subtraction of 2k from (1) yields

$$N = 2^k \left(\sum_{i=0}^{k} 2^{i-k} x_i \right) = 2^k \times X \qquad (2)X_A \times X_B. \qquad (3)$$

where X is an integer fraction between 1.0 and 2.0 (2) may be used to determine the outcome of multiplying A by B.

Because XA and XB have the same width as A and B, calculating XA and XB is quick and resource-intensive. We suggest determining this term's estimated value using the fractional portions of XA and XB. In the rest of this work, the fractional portion of X is represented as Y derived from $Y = X - 1.$ (4)

Consider the situation when X = (1.1101) 2. Y equals X in this scenario (0.1101) 2. We split the range (0.0–1.0) into S equal segments to compute Y, where S is a power of two denoted by.

$$S = 2^h \qquad (5)$$

h is a random positive integer that we use as one of our design factors. Each segment's length is plainly equal to 1/S. We suggest to construct an approximation of Y's value as

$$Y_{\mathrm{APX}} = \frac{2m-1}{2S} \quad \text{if } \frac{m-1}{S} \leq Y < \frac{m}{S}, \quad m = 1, 2, \ldots, S. \qquad (6)$$

ISSN: 2278-4632
Vol-12 Issue-06 No.03 June 2022

Fig. 1 depicts the estimated quantities of Y for the instance when S equals 4 to provide a better illustration. To find YAPX, just the h most relevant parts of Y must be taken into account.

When S = 4(h = 2), for example, the two most important bits of Y are zero, this indicates that 0 Y 1/4.

Consequently, we select 1/8 = (0.001)2 as YAPX. When the Two of Y's most important bits are "10," indicating that 2/4 Y 3/4, YAPX is roughly equivalent to 5/8 = (0.1012). To put it another way, YAPX is created by truncating Y to h bits and adding a "1" bit. Consequently, YAPX will have a width of h + 1 bits. Utilizing (4), (3) is rewritten as

$$A \times B = 2^{k_A+k_B} \times (1 + Y_A + Y_B + Y_A \times Y_B). \qquad (7)$$

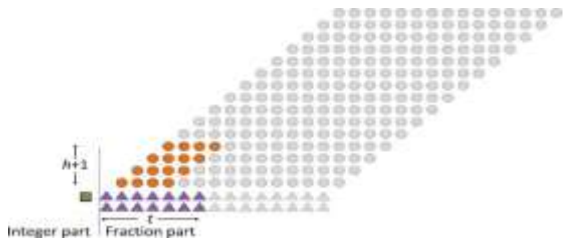


Fig.1. Dot diagram of term 1+(YA)t +(YB)t +(YA)APX×(YB)APX where t = 7 and h = 3.

Now, the approximate value of (7) may be written as

$$A \times B \approx 2^{k_A+k_B} \times (1 + Y_A + Y_B + (Y_A)_{\mathrm{APX}} \times (Y_B)_{\mathrm{APX}}). \quad (8)$$

To enhance computation performance, we truncate YA and YB to t bits, which we will refer to throughout this study as (YA)t and (YB)t. Consequently, we amend (8) as follows:

$$A \times B \approx (A \times B)_{\mathrm{APX}} = 2^{k_A+k_B} \times \begin{pmatrix} 1 + (Y_A)_t + (Y_B)_t + \\ (Y_A)_{\mathrm{APX}} \times (Y_B)_{\mathrm{APX}} \end{pmatrix} \qquad (9)$$

where (YA)APX ((YB)APX's width is h + 1 bits. To aid clarity, Fig. 2 compares the suggested method's dot diagram for the case where t = 7 and h = 3 to that of an accurate 16 -bit multiplier. The "1" in the formula 1+(YA)t +(YB)t + (YA)APX (YB)APX is represented by the green square. Orange circles represent the partial products of (YA)APX(YB)APX, Purple triangles indicate the component portions of (YA)t and (YB)t, respectively. Gray circles and triangles are excluded and disregarded from the computations. As illustrated in Fig. 2, the eventual outcome of the accurate 16-bit multiplier needs the inclusion of 256 partial products, while the suggested technique preserves just 31 partial products. As the bit length of the multiplier's input operands rises, this reduction rate will increase. Fig. 3 illustrates the methods for multiplying A by B when t = 7 and h = 3 as an example. In the remainder of this study, our suggested structures are denoted by TOSAM (X, Y), where X and Y correspond to h and t.

Based on factors t and h, a proposed method's accuracy may vary greatly. To attain near-high precision with acceptable speed and power, t and h will be related. Finally, Multiplying unsigned operands is recommended. C1 must determine the absolute value of the input operands, multiply them using the given approach, and then adjust the sign of the final output. Absolute value calculations may delay down. This necessitates the employment of the approach outlined in.

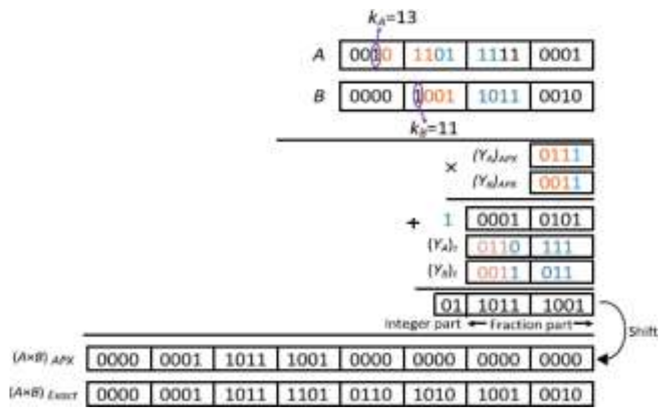


Fig.2 A = 11761 and B = 2482 is a 16-bit TOSAM(3, 7) example using A = 11761 and B = 2482. The approximate result [(A B)APX] is 28 901 376, but the precise result [(A B)Exact] is 29 190 802. In this instance, the absolute error is 289 426, which corresponds to around 0.99 percent of the actual result (theerror is less than 1 percent in this case).

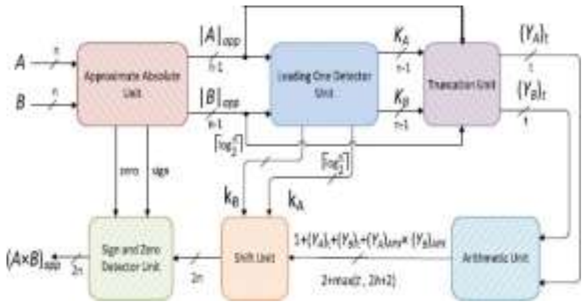

Approximate Absolute Unit
Leading One Detector Unit
Truncation Unit
Sign and Zero Detector Unit
Shift Unit
Arithmetic Unit

Fig.3. Block diagram of the proposed approximate signed multiplier.

Hardware Implementation

The suggested signed approximation multiplier's block diagram is shown in Figure 4. The Approximate Absolute Unit, a method similar to, is used to determine the approximate absolute value of the input operands (|A|app, |B|app). If the input is negative, the bits of this unit are reversed; otherwise, they remain unchanged. The Leading-One Detector Unit receives |A|app and |B|app, and the algorithm calculates the locations of their leading one bits.

If I might be either application A or application B Because just one bit of the signal K is "1," it signifies that the input location has been advanced by one bit. kA and kB signals may be produced using a lookup table using KA and KB signals (7). Figure 3 shows the Leading-One Detector Unit's 8-bit input design. Imagine this: |A|app = (011001)2, KA = (010000)2, kA = (100)2 = 4. The Truncation Unit receives |A|app, |B|app, KA, and KB to produce (YA)t and (YB)t. Input and output are I and (Y)t.

The Arithmetic Unit then solves 1 + (YA)t + (YB)t + (YA)APX (YB)APX using (YA)t and (YB)t. The rightmost bits of (YA)t and (YB)t are always "1," just like (YA)APX and (YB)APX. As a consequence, no additional hardware is required to generate (YA)APX and (YB)APX signals since they may be generated with simple wiring.

accuracy levels of TOSAM

The Arithmetic Unit's output is moved to the left by kA + kB by the Shift Unit. This makes

ISSN: 2278-4632
Vol-12 Issue-06 No.03 June 2022

the term 2kA + kB (1+(YA)t + (YB)t + (YA)APX (YB)APX). [Also see (9)] Sign and Zero Detector Unit output sign is determined by multiplier input operands. If one input is zero, so is the output. Approximate Absolute Unit, Sign and Zero Detector Units must be changed for unsigned multipliers.Figure 6 shows how the partial product drop levels change depending on how the machine is running. In the last level, a fast 9 -bit adder is used. Depending on the operating mode, the inputs of a transmission gate (TG) are set to "0" to slow down switching.

In T2 mode, only purple partial products are gathered and all 9 -bit adder inputs are "0." The operator is used to set the 10 least-important bits of the result to "0."

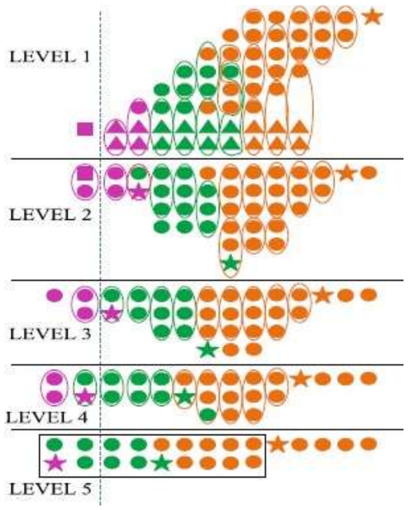


Fig. 4. TOSAM has three separate operation modes with customizable accuracy levels. Transmitting TGs and purple stars creates the four most important output bits. In T6 mode, only the green and purple partial products are created. Orange adders are powered by a power gate. Orange adder inputs also get "0" In T6 mode, two orange circles in LEVEL1's eighth column should be "0." Six least-important bits are set to "0." Green stars produce four intermediate output bits, and a 9-bit adder delivers the four most critical bits. In T9 mode, every component is turned on. The TGs process the orange stars to produce the output's five least significant bits, while a 9-bit adder creates the remaining bits. Depending on the operating mode, (Y A)APX and (Y B)APX should be rounded (set to "1"). Simply OR the operating mode bit. Logical OR sets (Y A)APX and (Y B)APX to "1" when T6 is "1."

Carry Carry Look Ahead Adder

Carry-look-ahead adders (CLA) are digital logic adders. Carry-look ahead adders improve performance by minimising time spent locating carry bits. The carry bit and the sum bit are computed simultaneously in the ripple-carry adder (RCA). Before starting its own sum bit and carry bit computations, each stage must wait for the previous stage's carry bit to be computed. One or more carry bits are calculated in the carry-look-ahead adder before calculating the total number of digits. This makes it faster to figure out the result of the adder's higher-value bits. The first phase is defined by the variables carry generate Gi and carry propagate Pi.

ISSN: 2278-4632
Vol-12 Issue-06 No.03 June 2022

$P_i = A_i \oplus B_i$

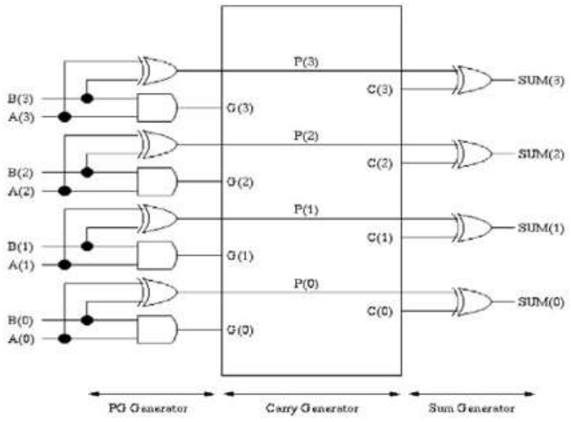
$Gi = Ai\ Bi$

Then the next step is carry generation

$Ci + 1 = Gi + Pi\ Ci$

The last step is post processing sum output t can be expressed as

$Si = Pi \oplus Ci$



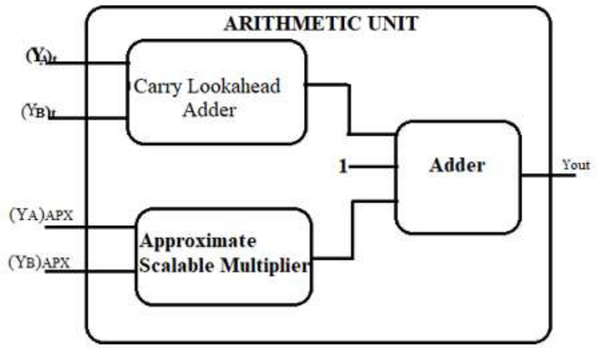


Fig 5: bit carry look ahead adder Fig 6: proposed arithmetic unit

4. Results

RTL SCHEMATIC: The register transfer level (RTL) schematic denotes the architecture's blueprint and is used to compare perfect architecture that we must create from the intended architecture. The hdl language is used to transform the architecture's description or summary into the functioning summary using a coding language like verilog or vhdl. The internal connection blocks are even specified in the RTL schematic for easier analysis. Below is a schematic representation of the design's RTL implementation.

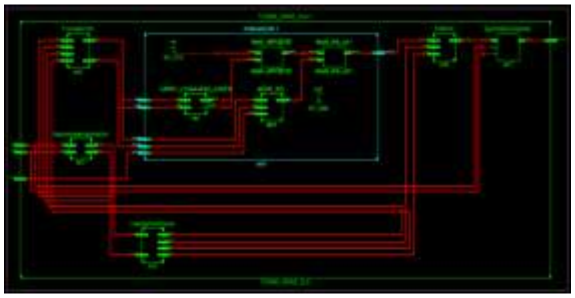


Fig7: RTL schematic of proposed TOSAM

TECHNOLOGY SCHEMATIC: With the LUT area parameter being used to estimate architecture design in VLSI, this diagram shows the technology's architecture in LUT format. The FPGA's LUTs, which are square units, represent the code's memory allocation.

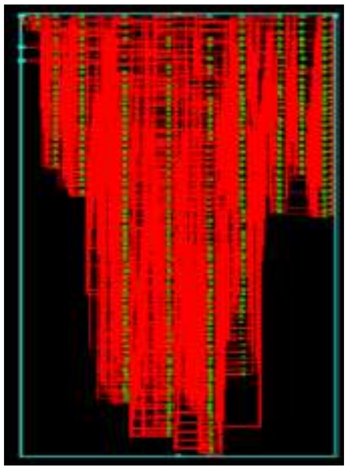


Fig8: view technology schematic of proposed TOSAM

SIMULATION:-Unlike the schematic, which only verifies the connections and blocks of a circuit, a simulation verifies the circuit's workings. Waveforms are the only form of output that can be viewed in the simulation window because it is launched by shifting from implementation to simulation. Since it can support multiple radix number systems, this is a useful feature.

Fig 9: Simulated waveforms of proposed TOSAM

PARAMETERS: Consider that in VLSI, the factors considered are area, delay, and power; using these metricsIt's possible to make comparisons between several designs. XILINX 14.7 is used to acquire the parameter, while verilog is used as the HDL language.

Table1: parameter comparison table

Parameter	TOSAM-CSLA	TOSAM-CLA
No of LUTs	370	357
Power (m.Watt)	2.15833	2.0825
Delay (ns)	27.721	26.264

GRAPH: The graph is a graphical representation of the represented data, allowing for easy comparative estimation. This graph depicts a comparison between the latency of two designs on a nanosecond scale.

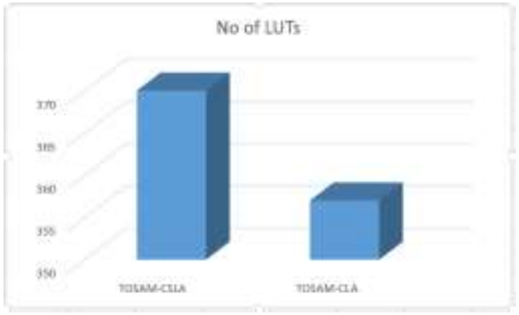


Fig10: lut comparison bar graph

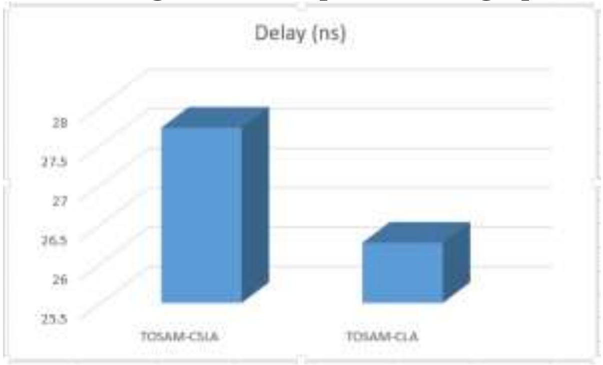


Fig11: delay comparison bar graph

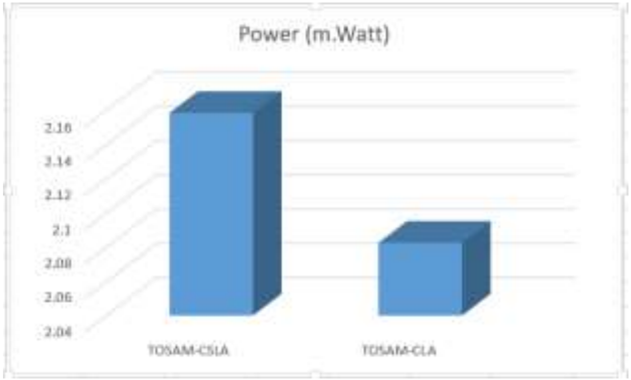


Fig12: power comparison bar graph

CONCLUSION

In this project, we presented a high-speed approximation multiplier with input operands trimmed to two unique lengths, t and h. The proposed multiplier outperformed current approximation in terms of how fast they work.The suggested 16-bit multiplier meets the speed requirements. Upon implementation of the suggested design, the space decreases from 370 to 357 square feet. Therefore, the suggested design is constructed and simulated using XILINX14.5 ISE and verilog HDL language, and the parameters are observed in spartan6 lowpower. Also, the suggested multiplier's great precision makes it an excellent candidate for use in image processing and classification applications. This paper proposes a high-speed, truncated and rounding-based, scalable 16-by-16 multiplier that is suitable for floating-point numbers. We use TOSAM algorithm to obtain high speed. This approximate computing is a calculation approach that delivers a potentially erroneous and it may be used in instances when an estimate is sufficient rather than a guaranteed correct solution. In the future, these multipliers will be used in the majority of designs where precision is not required, such as image processing filters, cryptography, etc.

REFERENCES

[1] 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.
[2] S. Xu and B. C. Schafer, "Exposing approximate computing optimizations at different levels: From behavioral to gate-level," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 11, pp. 3077–3088, Nov. 2017.
[3] A. Raghunathan and K. Roy, "Approximate computing: Energyefficient computing with good-enough results," in *Proc. IEEE 19th Int. On-Line Test. Symp. (IOLTS),* Chania, Greece, Jul. 2013, p. 258.
[4] D. Jeon, "Energy-efficient digital signal processing hardware design," Ph.D. dissertation, Dept. Elect. Eng., Michigan Univ., Ann Arbor, MI, USA, 2014.
[5] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "LETAM: A low energy truncation-based approximate multiplier," *Comput. Elect. Eng.*, vol. 63, pp. 1–17, Oct. 2017.
[6] 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, Nov. 2015, pp. 418–425.
[7] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBa multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 393–401, Feb. 2017.
[8] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 421–430, Mar. 2018.
[9] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 4, pp. 1352–1361, Apr. 2017.
[10] M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Syst. Lett.*, vol. 10, no. 1, pp. 6–9, Mar. 2018.