AMCAL: APPROXIMATE MULTIPLIER WITH THE CONFIGURABLE ACCURACY LEVELS FOR IMAGE PROCESSING AND CONVOLUTIONAL NEURAL NETWORK

Rana Fatima[1], Dr. M. Pavithra Jyothi[2]
[1]PG Scholar, Department of VLSI, Shadan Women's College of Engineering and Technology, Hyderabad, ranafatima2485@gmail.com
[2]Associate Professor, Department of ECE, Shadan Women's College of Engineering and Technology. m.pavithrajyothi@gmail.com

ABSTRACT

In machine learning algorithms and signal processing, multiplication is an essential arithmetic operation. In order to achieve energy and space reductions, this research suggests a unique technique called Approximate Multiplier with Configurable Accuracy Levels (AMCAL). The suggested approach manipulates the leftover bits and truncates the least significant bits to carry out the multiplication operation. The remaining input bits have been manipulated using a smaller bit width in order to minimize or eliminate the error that results from truncating the bits. Additionally, it has demonstrated that, in contrast to other algorithms, we are less concerned with the size of the mistake introduced into the input. When compared to the Wallace multiplier, the suggested AMCAL multiplier reduces energy usage by 86.7%, with an error margin of 0.15%. Furthermore, the suggested multiplier performs better than other approximate multipliers in the same class, including DRUM and DSM, in terms of latency, power, and area. In terms of power and area efficiency, the AMCAL multiplier outperforms modern approximation multipliers like DSI and TOSAM. Lastly, it is demonstrated that the picture quality produced by four image processing programs—smoothing, sharpening, JPEG encoding, and face alignment—is unaffected by such a little computational mistake.

INTRODUCTION

Enhancing the functionality and lowering the power consumption of digital circuits have gained more attention as a result of the rapid expansion of portable electronics like smartphones and other gadgets. However, the processors in these devices' cores feature computational units that are essential to carrying out various tasks.

One technique to improve efficiency and lower the power consumption of computing units is to use approximation computation for arithmetic operations. Because certain applications may tolerate erroneous results, the idea of approximation computation was born. To put it another way, these calculations may be used to programs that are robust against errors while they are operating. For instance, [1] mentions that certain devices exhibit error resistance and are insensitive to input accuracy. According to the authors in [2], approximation circuits can be used in programs when the mistake is recognized in a region that is unintelligible to humans, such wireless communications or video and audio programs. A certain amount of approximation won't be too harmful because digital signal processing devices already get a lot of noise from their surroundings, according to [3]. Additionally, approximation circuits that allow the computational blocks to adjust their accuracy level based on the output's needed precision have been provided by [4] and [5].

From the algorithm level to the transistor level, digital circuit designers have been working to decrease the space, power, and latency of processing blocks in recent years. In recent years, algorithms in artificial intelligence and neural networks [6]–[8] have presented power issues because of their high processing demands, opening the door for the inclusion of approximations in these programs. Since multiplications account for the biggest portion of all operations in the arithmetic logic unit (adders, multipliers, and dividers), which forms the basis of computing blocks, the approximation multiplier is the main focus of this work [9]. Multipliers are a desirable option for enhancing design characteristics with approximation circuits because of their high space and power consumption costs.

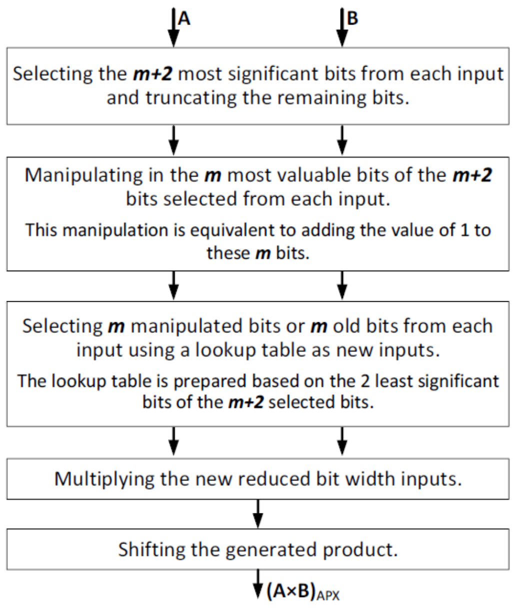

A
B
Selecting the m+2 most significant bits from each input and truncating the remaining bits.
Manipulating in the m most valuable bits of the m+2 bits selected from each input.
This manipulation is equivalent to adding the value of 1 to these m bits.
Selecting m manipulated bits or m old bits from each input using a lookup table as new inputs.
The lookup table is prepared based on the 2 least significant bits of the m+2 selected bits.
Multiplying the new reduced bit width inputs.
Shifting the generated product.
(A×B)APX

Fig 1. Overall flow diagram of AMCAL multiplier model.

Four categories may be used to group computational circuit approximation approaches, according to [10]:
1. Voltage scaling that is aggressive [11], [12]
2. Bit width truncation [5, 13, 14].
3. Making use of rough construction pieces [15], [16]
4. Stochastic computing usage [17], [18]

In order to lower energy usage, Chippa et al. [11] and Zervakis et al. [12] looked at hardware and architecture for aggressive voltage scaling. By truncating bit width and rounding multiplier inputs, Vahdat et al. [5] devised a technique that drastically decreases power and area in exchange for an output inaccuracy in the range of 0.3 to 11%. A 4:2 compressor was created by Ansari et al. [15] and Zakian et al. [16] in order to build two 4×4 multipliers with varying accuracies. They then utilized them as the fundamental components of 32x32 and 16x16 multipliers. Stochastic computing (SC), an unorthodox computation technique that considers data as probabilities, is used by Alaghi et al. [17]. SC is very tolerant of soft faults and makes advantage of massively parallel systems. Low precision, sluggish processing, and intricate design requirements are some of its disadvantages.
The fact that multiplication algorithms apply the identical operation to both inputs is one of its drawbacks. Using this method may result in substantial input mistakes that cause considerable output errors. An error detection and correction block may solve the problem, but it would increase the latency and power consumption. [5, 19, 20]. In contrast to DSM [19] and DRUM [20], we are less concerned with input mistakes in this article. Regardless of the magnitude of the mistake, the design has been developed to ensure that its impact on the output error is, for the most part, negligible or neutral. For instance, when the two values are multiplied, the mistake in the output is 0.05%, meaning that the effect of the error is 36 times lower, even if the input error was 1.8%. The output error for DSM4 and DRUM4 is 7.2% and 2.1%, respectively, given these inputs.
Additionally, working with beginning bits is inefficient because the estimated value will be displayed at the output. Therefore, it is possible to reduce the approximation effects on the output by adjusting the input bits.

Our suggested algorithm's competitive edge is that it has been much enhanced in terms of performance since the remaining bits of the inputs compensate for the impacts of the shortened bits without the need for extra error detection and correction blocks area, power, and delay. Bit truncation serves as the foundation for the multiplication operation in the suggested approach. In this manner, the most significant bit equal to "1" is chosen from each n-bit input, and the remaining bits are truncated. The two m-bit and 2-bit portions of these m+2 bits are then separated. Ultimately, the remaining two least significant bits are used to alter the m most significant bits from the selected m+2 bits to create m' bits.
Due to this modification, the mistakes of individual inputs are not added together when these m' bits from input A are multiplied by m' bits from input B; rather, they neutralize each other's faults. In contrast, the characteristics of power, area, and delay are greatly enhanced due to $m \ll n$. The AMCAL multiplier model's overall flow diagram is seen in Fig. 1.

LITERATURE SURVEY

This section provides a quick overview of some of the earlier research in the area of approximation multipliers. Two approaches were presented by Parekh et al. [13]. The first approach selects a few bits from the most important input bits and then performs multiplication, based on the idea that it is not essential to use all of the input bits to multiply roughly. The number of bits chosen from the inputs is regarded as floating in the second approach. The secondary input bits are chosen based on the number of primary input bits that are occupied. Lastly, they assert that 32-bit and 64-bit multipliers have much lower power and area. An approximate 4:2 compressor was created by Ansari et al. [15] in order to build two 4×4 multipliers with different levels of precision. They then demonstrated a 44% improvement in the product of power and latency by using them as building pieces for 16×16 and 32×32 multipliers. Lastly, for the first time, they looked at the performance of their suggested multiplier in MIMO antenna communication systems.
Stochastic computing is used by Alaghi et al. [17] to handle data as probabilities. Conventional logic circuits are used to create and analyze N-bit stochastic numbers (SNs), where each bit is typically randomly selected to be 1 with a probability PX. They conclude by looking at stochastic computing in terms of accuracy and mistakes, expenses associated with size and speed, design concerns, circuit-level elements, and applications.

The Truncation and Rounding-Based Scalable Approximate Multiplier (TOSAM), which was proposed by Vahdat et al. [4], multiplies the remaining bits by a sequence of adds and shifts. Their results, which are based on modeling and synthesis in 45 nm technology, show that an output error of 0.3–11% may be obtained in exchange for a 98%, 90%, and 41% reduction in power, area, and latency, respectively. The DSI approximate multiplier, a customizable approximate multiplier based on the idea that each incoming bit may be split into three parts—direct, search, and ignore—was recently developed by Hajizadeh et al. [5]. They showed that their approach, which involves operand swapping and the insertion of the Ex. bit, significantly reduced the average mistake rate when compared to earlier methods.

The DSM multiplier was first presented by Narayanamoorthy et al. [19] by truncating the least significant bits, greatly lowering the input bit width, and enhancing power, area, and delay. Like DSM, DRUM was introduced by Hashemi et al. [20]. To increase accuracy, it truncates the least significant bits and adds a bit with the value "1" to the end of the truncated bits before they reach the multiplier.

The AMCAL multiplier circuit will be contrasted with DSM [19] and DRUM [20] multipliers to illustrate its benefit, as the construction of the multiplier suggested in this study is likewise predicated on truncating the least important bits. Furthermore, we will contrast our suggested multiplier with two novel multipliers that employ an algorithm distinct from AMCAL: TOSAM [4] and DSI [5]. Additionally, we contrast the sign multiplication mode with TOSAM, DRUM, and ROBA [21] multipliers.

Several approximation designs for calculating the FFT are shown in this work. The word length is changed at each computing step to accomplish the balance between accuracy and performance. There are two suggested algorithms for changing the word length within a certain error margin. In contrast to the traditional fixed design, the first method aims for an approximation FFT for an area-limited design; the second approach focuses on performance in order to reach a higher operating frequency. Both of the suggested techniques demonstrate that it is feasible to achieve an effective stage-level balance between performance and hardware consumption. Experimental findings demonstrate that the first approximation technique reduces hardware consumption by at least almost 40% when the suggested approximate FFT designs are implemented on FPGA. The second method improves the designs' performance by more than 20%. In comparison to a coarse design, the FPGA resources required for a 256-point FFT calculation may be further decreased by over 10% with fine granularity design, which is currently being studied. Lastly, a feature extraction module in a separate word recognition system is implemented using the suggested approximate designs; the Mel frequency cepstrum coefficients (MFCC) extraction module's LUT and FF counts are reduced by up to 47.2% and 39.0%, respectively, with a power reduction of up to 27.0% at a less than 2% accuracy loss.

The truncation-and-rounding-based scalable approximate multiplier (TOSAM) is a scalable approximate multiplier that truncates each input operand according to its leading one-bit location, therefore reducing the number of partial products. In contrast to the precise multiplier, the suggested architecture significantly reduces energy consumption and area occupied by performing multiplication using shift, add, and short fixed-width multiplication operations. The multiplication part's input operands are rounded to the closest odd value in order to increase overall accuracy. The multiplier becomes scalable and the precision becomes weakly reliant on the width of the input operands due to the truncation of the input operands depending on their leading one-bit locations. As the input operand widths rise, more improvements may be made to the design characteristics (such as area and energy usage). The suggested approximate multiplier's effectiveness is assessed by contrasting its design parameters with those of an exact multiplier and a few other recently suggested approximate multipliers. The results show that when compared to the precise multiplier, the suggested approximate multiplier with a mean absolute relative error between 11% and 0.3% reduces latency, area, and energy usage by up to 41%, 90%, and 98%, respectively. Additionally, it performs better than other approximation multipliers in terms of energy usage, speed, and area. The error distribution of the suggested approximation multiplier is almost Gaussian, and its mean value is close to zero. We take advantage of it in applications for categorization, sharpening, and the structure of a JPEG encoder. The output's quality deterioration is minimal, according to the results. Furthermore, depending on the minimum needed accuracy, we propose an accuracy customizable TOSAM in which the energy consumption of the multiplication operation may be modified.

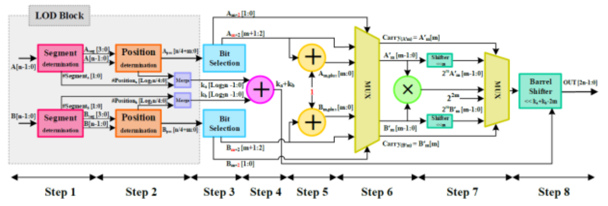
These days, machine learning algorithms and automation techniques are essential in practically every area. In this study, the probabilistic pruning approach is used to mimic a 4:2 compressor circuit. For the suggested 4:2 compressor, an artificial neural network is created and trained to achieve the test and train accuracies. The best approximation circuit has been thought to be a neural network with equal train and test accuracies. Using the truth table of the suggested approximate 4:2 compressor as the dataset, a supervised machine learning approach was used to train the neural network. With just 19 transistors, the suggested compressor uses less energy (0.2015 nJ) and has a smaller silicon area (14.36 um2). By substituting the suggested approximate 4:2 compressor into its partial product reduction step, the Dadda multiplier's performance is enhanced.

For portable multimedia devices using different signal processing methods and architectures, low power consumption is a crucial need. Humans can infer valuable information from somewhat inaccurate outputs in the majority of multimedia applications. Consequently, we are not required to generate numerical results that are precisely accurate. Prior work in this area mostly uses voltage overscaling to leverage error resilience, mitigating the ensuing faults with computational and architectural strategies. In order to exploit the relaxation of numerical precision, we suggest in this study a different strategy: logic complexity reduction at the transistor level. In order to illustrate this idea, we suggest a number of approximate or imprecise complete adder cells that are simpler at the transistor level. We then use these cells to create approximation multi-bit adders. Apart from the intrinsic decrease in switching capacitance, our methods lead to noticeably shorter critical pathways, which facilitate voltage scaling. Using the suggested approximate arithmetic units, we create designs for image and video compression algorithms and test them to show how effective our method is. Additionally, we establish basic mathematical models for these approximation adders' inaccuracy and power consumption. Additionally, we show how useful these approximation adders are in two digital signal processing systems with particular quality constraints: the finite impulse response filter and the discrete cosine transform. Comparing the suggested approximation adders to current implementations that use correct adders, simulation findings show power savings of up to 69%.

Support for diverse DSP and classification applications on energy-constrained devices has become more important. These applications frequently use fixed-point arithmetic to execute matrix multiplications in large quantities while displaying tolerance for certain computational faults. Therefore, increasing multiplications' energy efficiency is essential. In this short, we provide multiplier topologies that allow for the trade-off between energy usage and computational accuracy throughout design. With an average computational error of about 1%, the suggested multiplier can use 58% less energy per operation than a precise multiplier. Lastly, we show that the accuracy of classification applications and the quality of DSP are not significantly affected by such a little computational mistake.

PROPOSED METHODOLOGY

BLOCK DIAGRAM



Block diagram showing how the AMCAL unsigned multiplier circuit is implemented.

The following is a summary of this paper's contributions:

- By altering the traditional multiplication method, presenting a new multiplication scheme.
- Restructuring the algorithm used to determine the leading bit's location to use less space and energy.
- Examining the lookup table to reduce the output error by figuring out the manipulation instructions in certain bits.
- Outlining three approximation multiplier hardware solutions for signed and unsigned operations that have adjustable accuracy levels (AMCAL).
- Examining the suggested multiplier's design settings for face alignment, JPEG image compression, and picture processing.

MODULE EXPLANATION:

A. AMCAL Multiplier Arithmetic for Multiplication

Certain current approximation multiplication algorithms work under the assumption that the input will never change; they either utilize an approximate adder, accept the approximate output and use it to approximate the building blocks of partial products, or they disregard some pieces of the input [4], [15]. Our suggested multiplication technique by small input variation makes it possible to reduce the number of input bits more effectively, which lowers the power, area, and latency of the multiplier. The fundamental idea of the suggested method is derived from (1):

$$\begin{aligned} A \times B &= (A \pm E_1)(B \pm E_2) \\ &= AB \pm BE_1 \pm AE_2 \pm E_1E_2 \end{aligned}$$

Where $E1$ and $E2$ represent the adjustments we made to the input values (which we will now refer to as input errors). To get a reasonable approximation, the terms $BI1$, $AE2$, and $E1E2$ must be eliminated. We used the following method to do this:

1- The word E1E2 can be eliminated from (1) if we choose input errors E1 and E2 so that E1E2

2. The phrase $\pm BE1 \pm AE2$ can be eliminated from (1) if we choose input errors E1 and E2 so that the terms ±BE1 and ±AE2 cancel each other out. In order to do this, input errors E1 and E2 must be directed in opposite directions (positive and negative, or vice versa), and they must, respectively, represent a proportion of inputs A and B (Equation (2)).

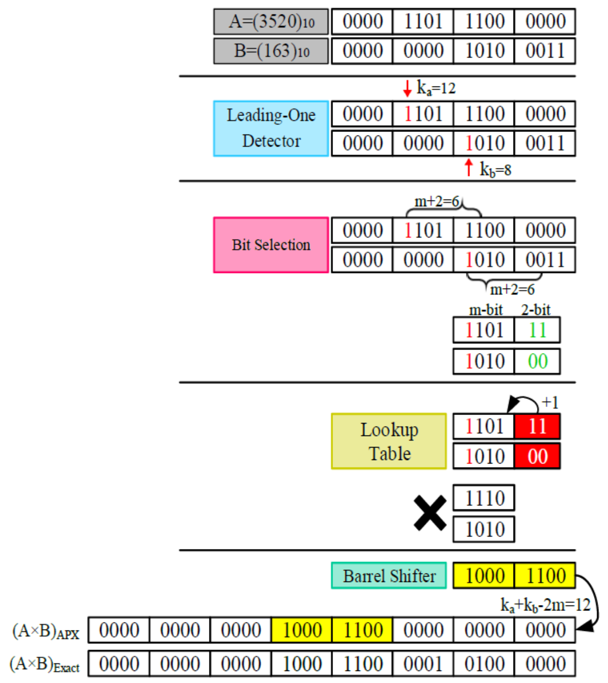
$$\begin{aligned} E_1 = E_A = \%x_1 \times A \\ E_2 = E_B = \%x_2 \times B \end{aligned}$$

When we replace (2) with (1), we obtain:

$$\begin{aligned} A \times B &= (A \pm E_A)(B \mp E_B) \\ &= AB \pm BE_A \mp AE_B \pm E_AE_B \\ &= AB \pm B\frac{x_1}{100}A \mp A\frac{x_2}{100}B \pm \frac{x_1}{100}\frac{x_2}{100}AB \end{aligned}$$

The words ±BEA∑AEB cancel each other out if the equivalence x1=x2=x is true. If x1,x2≥100, the term x1100x2100AB can be disregarded in relation to AB. Thus, (3) becomes (4).

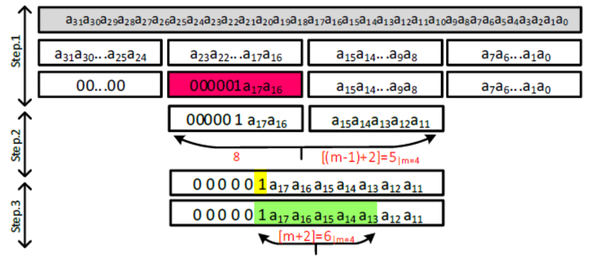
$$A \times B = \left(A \pm \frac{x_1}{100}A\right)\left(B \mp \frac{x_2}{100}B\right)$$
$$= \left(1 \pm \frac{x}{100}\right)\left(1 \mp \frac{x}{100}\right)AB$$
$$= \left(1 - \left(\frac{x}{100}\right)^2\right)AB \approx AB$$



An illustration of the AMCAL multiplication algorithm's steps for multiplying two integers is shown in FIG 2.

To minimize the space and power consumption, it is erroneous to select $x1 = x2$ since two n-bit additions must be done in addition to a single n-bit multiplication. In order to address this problem, $x1$ *and* $x2$ are chosen in a way that maximizes the satisfaction of the two aforementioned requirements while simultaneously improving the power consumption, area, and delay.
In an n-bit number, the weight at each bit location drops exponentially from the most significant bits to the least significant bits. Consequently, we may take the input value that contains just m most significant bits from the original input and truncate the n-m least significant bits using one of the two methods listed below:
1. Reduce the value of the input in comparison to its original input by eliminating the n–m least significant bits.
2. The sum of the remaining portion (the m most significant bits) plus "1" is computed to raise the number in relation to its starting value in order to restore the inaccuracy brought on by deleting the n–m bits.

Interestingly, truncating the least significant bits in the suggested approach is the same as adding an error to the input; this is comparable to the values of $x1$ and $x2$ in (2). Moreover, in order to meet the second criterion, one of the inputs needs to be larger than its original value and the other less than its initial value, meaning that E1 and E2 have the opposite signs.
As more selected bits (m) are chosen, less error is introduced into the inputs, resulting in lower values for $x1$ and $x2$. This lowers the values of the terms added in (3) and makes it simpler to disregard those terms.


Step.1
Step.2
Step.3
$a_{31}a_{30}a_{29}\ldots a_2a_1a_0$
$a_{31}a_{30}\ldots a_{25}a_{24}$
$a_{23}a_{22}\ldots a_{17}a_{16}$
$a_{15}a_{14}\ldots a_9a_8$
$a_7a_6\ldots a_1a_0$
00…00
$000001a_{17}a_{16}$
$000001\ a_{17}a_{16}$
$a_{15}a_{14}a_{13}a_{12}a_{11}$
8
$[(m-1)+2]=5|_{m=4}$
$00000\,1\,a_{17}a_{16}a_{15}a_{14}a_{13}a_{12}a_{11}$
$[m+2]=6|_{m=4}$

The implementation of the 32-bit LOD and Bit-selection module for a=00000000|000001xx|xxxxxxxx|xxxxxxxx when m=4 is shown in FIGURE 3.

The suggested multiplication method is seen in Fig. 2. essentially converting the 16-bit inputs to 4-bit values by manipulating them when m=4. Additionally, input B is less than its starting value while input A increases in value. As a result, the algorithm's requirements are all met.

B. Hardware Implementation of AMCAL Multiplier

The suggested multiplier circuit uses the method shown below, which is schematically shown in Fig. 4:
(1) The Leading-One Detector (LOD) Block receives the n-bit inputs. Subsequently, the segment determination block splits the n-bit input into four segments, each containing a bit equal to "1" (n/4) bits. Next, the priority encoder is used to create that segment's index.
(2) To create a n/4+m+1 bits number, the relevant (n/4)-bit component is combined with m+1 bits from the right-hand segment. The leftmost bit that equals "1" in the chosen segment is then located by the position determination block. Next, the priority encoder is used to produce that position's index.
(3) The (n/4+m+1)-bit inputs are sent to the Bit Selection block, which truncates the remaining bits and chooses the m+2 bits from the leftmost bit equal to "1" found in the previous stages (see to Fig. 3).
(4) By combining the #Segment and #Position priority encoder outputs from each input, the values of ka and kb are obtained. The amount of shift in the output is then computed by summing these two figures.
(5) An adder is entered with the m-bit integer to be added to "1".
(6) The Multiplexer block decides whether the m-bit number should be approximated higher (by incrementing

it by "1") or stay constant by taking the two least significant bits from the m+2 bits.
As seen in TABLE 1, the control bits for the Multiplexer are acquired by consulting a lookup table. The estimated input is then fed into a m×m multiplier after that.
(7) Based on the carry bits of the modified inputs A'm and B'm, a multiplexer chooses one of its four inputs. These four options are:
1 is the product of A'm and B'm,
2 is the shift of the input A'm left by m bits,
3 is the shift of the input B'm left by m bits, and
4 is the number 22m.
(8) A barrel shifter is used to show the produced product as a 2n-bit integer.

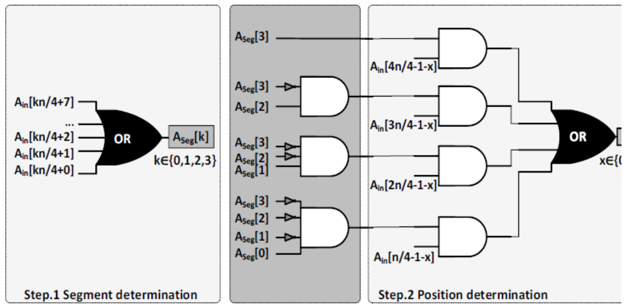


Fig 5. The Bit-selection block and the LOD block components

Since fewer AND and OR gates are required to perform the LOD and Bit-selection block in this scenario than in the case of splitting into 2, 8, or 16, four was the number four chosen to divide the inputs in step 1. We require the number of two-input OR gates given in (5) in order to construct the LOD and Bit-selection block in our suggested algorithm:

$$Total\ OR_2 = (s)\left(\frac{n}{s}-1\right)|_{segment\ determination} + \left(\frac{n}{s}+m+1\right)(s-1)|_{Position\ determination} + (m+2)\left(\frac{n}{s}-1\right)|_{Bit\ Selction}$$

We also need the two-input AND gates in order to get the number shown in (6).

$$Total\ AND_2 = \frac{(s-1)s}{2}|_{segment\ Encoder} + \left(\frac{n}{s}+m+1\right)(s)|_{Position\ determination} + \frac{\left(\frac{n}{s}-1\right)\left(\frac{n}{s}\right)}{2}|_{Position\ Encoder} + (m+2)\left(\frac{n}{s}\right)|_{Bit\ Selction}$$

In these Equations, *S* indicates the number of segments. Also, OR gates and AND gates with *k* inputs have been replaced with *k-1* two-input gates. By searching for all the available values for the variables s, n, and m, it can be concluded that the lowest number of gates when n equals 8 or 16 occurs at *s=4*. Also, when n equals 32, the lowest number of gates occurs at *s=8*. Although using *s=8* would use the lowest number of gates when operating in 32-bit mode, using *s=4* would use almost the same number of gates. As a result, we always divide the input into 4 parts, *s=4*, for integrity and independence from the number of input bits. Note that the value of m has no bearing on the value of s for the lowest number of gates.

C. Accuracy of AMCAL Multiplier

1) Parameters for measuring circuit performance:

The accuracy of diverse circuits is now faced with a new issue as a result of the excessive decrease of circuit complexity and delay. The many standards for assessing approximate circuit performance are covered in this section.

a) The mean error distance and error distance

The error distance (ED) is defined as the difference between the precise product M and the approximate product M', where $ED=|M'-M|$. The mean error distance (MED) is the average of all ED values for all inputs. The normalized mean error distance (NMED), where Mmax is the maximum product value of an exact multiplier, e.g., $(2n-1)2$ for an n×n multiplier, must be used to compare multipliers of different sizes.

b) Meaning of relative error and relative error

The relative error, represented by $RED = |M'-M|/M = ED/M$, may be utilized to more accurately compare different circuits. Averaging all of the REDs for various input combinations yields the mean relative error (MRED) for an n×n multiplier.

$$MRED = 2^{-2n}\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^n-1}\frac{|M'_{ij}-M_{ij}|}{M_{ij}}$$

Hence, given the inputs i = A and j = B, Mij' and Mij are the approximation and precise multiplications, respectively. Interestingly, when one of the inputs is 0, MRED is defined as zero.

c) Both The Pass And Mistake Rates

The ratio of accurate outputs to total outputs is known as the pass rate (PR), whereas the ratio of wrong outputs to total outputs is known as the error rate (ER).
The suggested model will also be assessed using mean squared error (MSE) and maximum error (Max ER), in addition to MRED, MED, NMED, and PR. Error will also be assessed using variance, error rate, and the Hamming distance (ACCinf) [23].
Lastly, the cost function is created in accordance with (10) to evaluate all parts of the algorithm simultaneously as the number of distinct chosen bits m increases, much like in [4]. It is preferable that the cost function have a smaller value.

$$CF=MRED\times Energy\times Delay\times Area$$

2) Parameters for measuring circuit performance:
As shown in (11), the maximum error is computed.

$$error(A,B)=\frac{(A_T-A)(B_T-B)}{AB}=(\frac{A_T}{A}-1)(\frac{B_T}{B}-1)$$

To get the greatest error, the terms $(AT/A-1)$ and $(BT/B-1)$ must be maximized. The bits to be truncated, the two least significant bits of the m+2 chosen bits, and the m most significant bits of the input that have their leftmost bit always equaling "1" must thus be separated into three sections. According to this classification, rows 1, 2, 3, and 4 of Table 2 display, respectively, the forms of inputs A and B that cause the maximum error, the number of times the maximum error occurs, the value of the maximum error, and the mean relative error when the inputs are 32-bit and the number of input bits selected is m=4.

D. Applications of Image Processing
When multiplier sizes change, error evaluation through simulation and programming becomes much more time-consuming. In error-resilient applications, the quality of the approximation multiplier may be evaluated using image processing techniques as picture smoothing and sharpening.
As seen in (12), the highest signal-to-noise ratio serves as another criterion for assessing the quality of the reconstructed image in comparison to the original image.

$$PSNR=10\log_{10}\frac{255^2}{\frac{1}{mn}\sum_{i=0}^{m}\sum_{j=0}^{n}[\alpha_{(i,j)}-\beta_{(i,j)}]^2}\ in\ dB$$

where the numbers α (i,j) and β(i,j) stand for the number of pixels in the original picture and the number of pixels in the reconstructed image at location (i, j) with dimensions m×n.
The picture smoothing procedure may be applied using equation (13), as described in [21], where i and j stand for the considered pixel.

$$Y(i,j)=\frac{1}{60}\sum_{m=-2}^{2}\sum_{n=-2}^{2}X(i+m,j+n).Mask(m+3,n+3)$$

Definition of the smoothing mask matrix used in [24]

$$Mask_{\text{smoothing}}=\begin{bmatrix}1&1&1&1&1\\1&4&4&4&1\\1&4&12&4&1\\1&4&4&4&1\\1&4&7&4&1\end{bmatrix}$$

An actual model of this 8-bit multiplier is created in MATLAB and used to smooth images in order to assess the suggested approach. It should be noted that the current simulation analyzes the effects of approximation multiplication by performing all other operations—such as addition and subtraction—exactly. Additionally, the MATLAB simulations of the DSM, DRUM, TOSAM, and DSI models are used in conjunction with the smoothing technique to quantify the quality of the picture output using the parameters provided in [25] for the peak-signal-to-noise ratio (PSNR) and mean structural similarity index metric (MSSIM). Lastly, TABLE 11 presents the findings.
In addition to TOSAM(1,5) and DSI2, which have the same error class as AMCAL3, it is compared to the DRUM4 and DSM5 models, which have four and five input bits, respectively, to highlight the benefit of AMCAL having just three input bits for the multiplication operation.
Image sharpening methods are another criteria for image processing testing, and (15), as described in [21], can be applied in this situation.

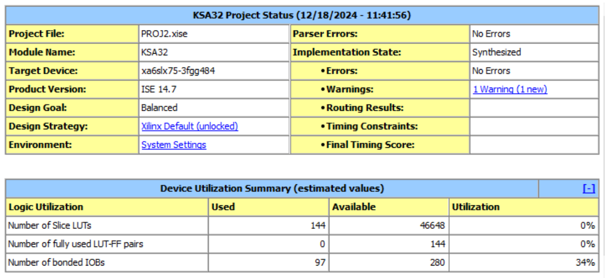
$$Y(i,j)=2.X(i+m,j+n)-\frac{1}{273}\sum_{m=-2}^{2}\sum_{n=-2}^{2}X(i+m,j+n).Mask(m+3,n+3)$$

(16) defines the sharpening mask matrix [24].

$$Mask_{\text{sharpening}}=\begin{bmatrix}1&4&7&4&1\\4&16&26&16&4\\7&26&41&26&7\\4&16&26&16&4\\1&4&7&4&1\end{bmatrix}$$

Lastly, the MATLAB model is subjected to the picture sharpening procedure in order to assess the quality arising from the different operating classes of AMCAL. From m=1 to m=8, eight operational classes are used to analyze the impacts. Figure 11 displays the final photos. As can be observed in the picture, the sharpening algorithm does a good job at sharpening the edges of the image in comparison to the original, and the decline in image quality drops significantly as AMCAL4 and higher.
In a different application, we used signed 16-bit approximation multipliers in the discrete cosine transform (DCT) unit and the JPEG method to compress several pictures. Processing DCT calls for a lot of addition and multiplication. The DCT precise multiplication is replaced by the approximate multipliers. To illustrate the benefit of S-AMCAL while performing the multiplication operation with just four input bits, we use the DRUM5 model, which has five input bits. Furthermore, because S-ROBA and TOSAM(2,6) belong to the same error class as S-AMCAL4, they are chosen for comparison.
Comparing the compressed pictures to the original input images, we were able to recover the PSNR and MSSIM, The outcomes demonstrate that AMCAL4 performs better in PSNR and MSSIM than its rivals.

RESULTS & DISCUSSION



KSA32 Project Status (12/18/2024 - 11:41:56)			
Project File:	PROJ2.xise	**Parser Errors:**	No Errors
Module Name:	KSA32	**Implementation State:**	Synthesized
Target Device:	xa6slx75-3fgg484	**•Errors:**	No Errors
Product Version:	ISE 14.7	**•Warnings:**	1 Warning (1 new)
Design Goal:	Balanced	**•Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	
Environment:	System Settings	**•Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	144	46648	0%
Number of fully used LUT-FF pairs	0	144	0%
Number of bonded IOBs	97	280	34%

Fig 1: Utilization Summary Report

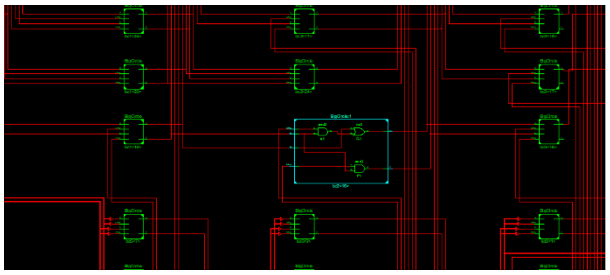


Fig 2: RTL Schematic (Gate-Level)

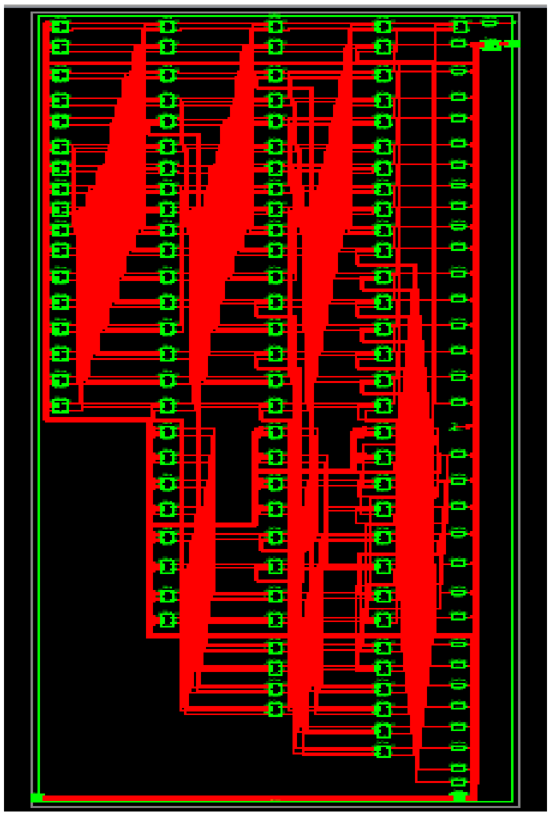


Fig 3: RTL Schematic (Gate-Level)

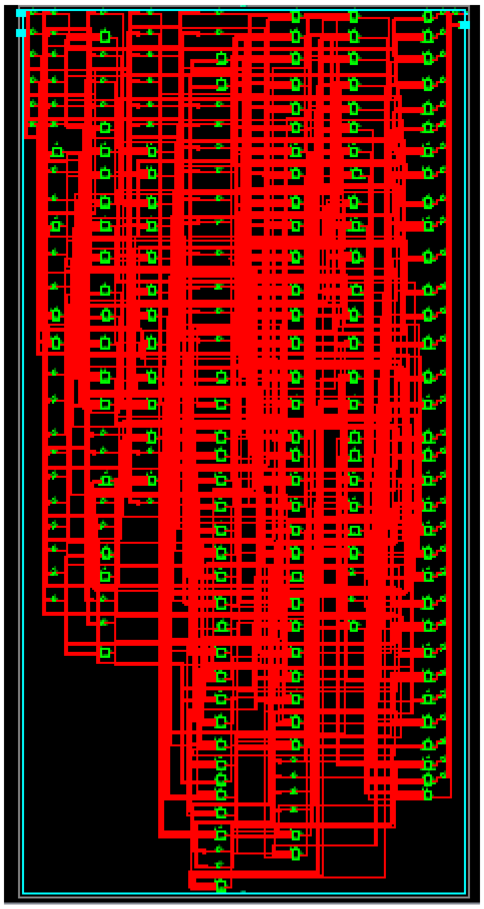
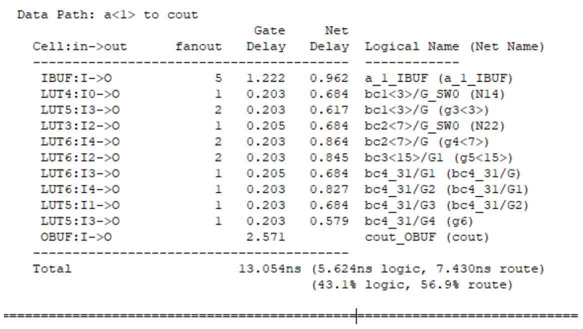


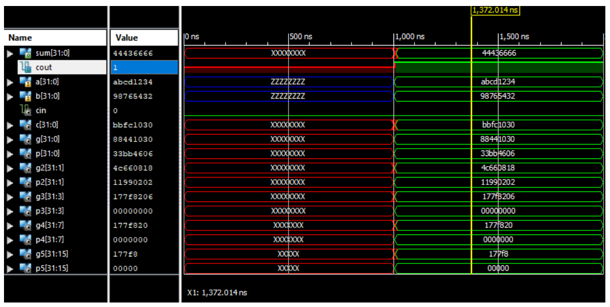
Fig 4. TTL Schematic



```
Data Path: a<1> to cout
                                Gate     Net
  Cell:in->out      fanout    Delay    Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O             5    1.222    0.962  a_1_IBUF (a_1_IBUF)
   LUT4:I0->O            1    0.203    0.684  bc1<3>/G_SW0 (N14)
   LUT5:I3->O            2    0.203    0.617  bc1<3>/G (g3<3>)
   LUT3:I2->O            1    0.205    0.684  bc2<7>/G_SW0 (N22)
   LUT6:I4->O            2    0.203    0.864  bc2<7>/G (g4<7>)
   LUT6:I2->O            2    0.203    0.845  bc3<15>/G1 (g5<15>)
   LUT6:I3->O            1    0.205    0.684  bc4_31/G1 (bc4_31/G)
   LUT6:I4->O            1    0.203    0.827  bc4_31/G2 (bc4_31/G1)
   LUT5:I1->O            1    0.203    0.684  bc4_31/G3 (bc4_31/G2)
   LUT5:I3->O            1    0.203    0.579  bc4_31/G4 (g6)
   OBUF:I->O                  2.571           cout_OBUF (cout)
  ----------------------------------------
  Total                      13.054ns (5.624ns logic, 7.430ns route)
                                      (43.1% logic, 56.9% route)

==============================================================================
```

Fig 5: Timing Summary



Name	Value
sum[31:0]	44436666
cout	1
a[31:0]	abcd1234
b[31:0]	98765432
cin	0
c[31:0]	bbfc1030
g[31:0]	88441030
p[31:0]	33bb4606
g2[31:1]	4c660818
p2[31:1]	11990202
g3[31:3]	177f8206
p3[31:3]	00000000
g4[31:7]	177f820
p4[31:7]	0000000
g5[31:15]	177f8
p5[31:15]	00000

Fig 6: Simulation Results

CONCLUSION

In the current work, an approximation multiplier known as AMCAL was presented. It is significantly more accurate than multipliers in the same class and requires less power and space. This multiplier operates on the premise of altering the input bits by lowering their number before to entering the main multiplier while preserving the range of error tolerance. You may use this multiplier for both signed and unsigned multiplication. We looked into eight different ways to apply this multiplier. By contrasting it with a number of precise and approximative multipliers that were created with varying design parameters, the suggested multiplier's effectiveness was assessed.

The results show that in all design parameters, the AMCAL architecture outperforms the Wallace exact multiplier, and in the majority of design parameters, it outperforms the DSM and DRUM approximation multipliers. The suggested multiplier has a larger delay parameter than the adjustable approximation multipliers TOSAM and DSI, but it performs better in terms of power and area parameters. Finally, four image processing programs—image sharpening, image smoothing, JPEG image compression, and face alignment—were used to test the suggested multiplier.

REFERENCES

[1] Gupta, Puneet, Yuvraj Agarwal, Lara Dolecek, Nikil Dutt, Rajesh K. Gupta, Rakesh Kumar, Subhasish Mitra, et al. "Under-designed and opportunistic computing in the presence of hardware variability." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 32, no. 1,8-23, 2013.

[2] Liu, Weiqiang, et al. "Approximate designs for fast Fourier transform (FFT) with application to speech recognition." IEEE Transactions on Circuits and Systems I: Regular Papers 66.12 (2019): 4727-4739.

[3] Venkatesan, Rangharajan, Amit Agarwal, Kaushik Roy, and Anand Raghunathan. "MACACO: Modeling and analysis of circuits for approximate computing." In Proceedings of the International Conference on Computer-Aided Design, pp. 667-673. IEEE Press, 2011.

[4] Vahdat, Shaghayegh, et al. "TOSAM: An Energy-Efficient Truncation-and Rounding-Based Scalable Approximate Multiplier." IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 27.5 (2019): 1161-1173.

[5] Hajizadeh, Fahimeh, et al. "Configurable DSI partitioned approximate multiplier." Future Generation Computer Systems 115 (2021): 100-114.

[6] Cardarilli, Gian Carlo, et al. "Approximated computing for low power Neural Networks." Telkomnika 17.3 (2019): 1236-1241.

[7] Zervakis, Georgios, et al. "Approximate Computing for ML: State-of-the-art, Challenges, and Visions." 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2021.

[8] Maddisetti, Lavanya, Ranjan K. Senapati, and J. V. R. Ravindra. "Accuracy evaluation of a trained neural network by energy efficient approximate 4: 2 compressor." Computers & Electrical Engineering 92 (2021): 107137.

[9] 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.

[10] 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.

[11] Liu, Bo, et al. "A Reconfigurable Approximate Computing Architecture With Dual-VDD for Low-Power Binarized Weight Network Deployment." IEEE Transactions on Circuits and Systems II: Express Briefs 70.1 (2022): 291-295.

[12] Zervakis, Georgios, et al. "Multi-level approximate accelerator synthesis under voltage island constraints." IEEE Transactions on Circuits and Systems II: Express Briefs 66.4 (2018): 607-611.

[13] Parekh, Prashil, Samidh Mehta, and Pravin Mane. "Truncation Based Approximate Multiplier For Error Resilient Applications." International Journal of Electronics Letters (2021): 1-12.

[14] Gu, Fang-Yi, Chao Lin, and Jia-Wei Lin. "A Low-Power and High-Accuracy Approximate Multiplier With Reconfigurable Truncation." IEEE Access 10 (2022): 60447-60458.

[15] Ansari, Mohammad Saeed, et al. "Low-power approximate multipliers using encoded partial products and approximate compressors." IEEE Journal on Emerging and Selected Topics in Circuits and Systems 8.3 (2018): 404-416.

[16] Sayadi, Ladan, Somayeh Timarchi, and Akbar Sheikh-Akbari. "Two Efficient Approximate Unsigned Multipliers by Developing New Configuration for Approximate 4: 2 Compressors." IEEE Transactions on Circuits and Systems I: Regular Papers (2023).

[17] Alaghi, Armin, Weikang Qian, and John P. Hayes. "The promise and challenge of stochastic computing." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37.8 (2017): 1515-1531.

[18] Najafi, M. Hassan, and Mostafa E. Salehi. "A fast fault-tolerant architecture for sauvola local image thresholding algorithm using stochastic computing." IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 24.2 (2015): 808-812.

[19] Narayanamoorthy, Srinivasan, et al. "Energy-efficient approximate multiplication for digital signal

processing and classification applications." IEEE transactions on very large scale integration (VLSI) systems 23.6 (2014): 1180-1184.
[20] Hashemi, Soheil, R. Bahar, and Sherief Reda. "DRUM: A dynamic range unbiased multiplier for approximate applications." Proceedings of the IEEE/ACM international conference on computer-aided design. IEEE Press, 2015
[21] Zendegani, Reza, et al. "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 (2016): 393-401.
[22] Nangate 45nm Open Cell Library. [Online]. Available: http://www.nangate.com/
[23] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in Proc. 49th Design Autom. Conf. (DAC), Jun. 2012, pp. 820–825.
[24] Myler, Harley R., and Arthur R. Weeks. The pocket handbook of image processing algorithms in C. Prentice Hall Press, 2009.
[25] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Trans. Image Process., vol. 13, no. 4, pp. 600–612, Apr. 2004.
[26] Jiang, Honglan, et al. "Approximate arithmetic circuits: A survey, characterization, and recent applications." Proceedings of the IEEE 108.12 (2020): 2108-2135.
[27] Zhang, Zhanpeng, et al. "Facial landmark detection by deep multi-task learning." European conference on computer vision. Springer, Cham, 2014.
[28] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[29] Watchareeruetai, Ukrit, et al. "LOTR: Face Landmark Localization Using Localization Transformer." IEEE Access 10 (2022): 16530-16543.
[30] Koestinger, Martin, et al. "Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization." 2011 IEEE international conference on computer vision workshops (ICCV workshops). IEEE, 2011.
[31] E. Bisong, Google colaboratory, in: Building Machine Learning and Deep Learning Models on Google Cloud Platform, Springer, 2019, pp. 59–64