International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

ADEPNET – A DYNAMIC –PRECISION EFFICIENT POSIT MULTIPLIER FOR NEURAL NETWORKS

Sandhya Rani Posam[1], Farina Yasmeen[2]
[1]PG Scholar, Department of VLSI, Shadan Women's College of Engineering and Technology, Hyderabad, sandhyavenkat126@gmail.com
[2]Asst. Professor, Department of ECE, Shadan Women's College of Engineering and Technology farinays@gmail.com

Received: 05-08-2025 Accepted: 07-09-2025 Published: 14-09-2025

ABSTRACT

The goal of the posit number system is to seamlessly replace the current IEEE floating point standard. When it comes to displaying decimals, its tapering accuracy and wide dynamic range enable a smaller size point to nearly equal the performance of a much larger size floating-point. When completing error-tolerant jobs where low latency and area are crucial, such as deep learning inference computation, this becomes quite helpful. According to recent studies, the precision of multipliers employed for convolutions reaches a point at which deep neural network models' performance saturates. As a result, creating accurate arithmetic circuits for these applications requires additional hardware, which is a needless expense. In order to strike the best possible balance between hardware use and inference accuracy, this research investigates approximation posit multipliers in the convolutional layers of deep neural networks. There are several phases in posit multiplication, with the mantissa multiplication step using the most hardware power. This is lessened by proposing a new multiplier circuit that uses a bit masking dependent on input regime size and an approximate hybrid-radix Booth encoding for mantissa multiplication. Furthermore, a new Booth encoding control mechanism has been developed to minimize dynamic power waste by preventing superfluous bits from switching. These changes have helped to reduce power dissipation in the mantissa multiplication stage by 23% as compared to previous research.

1. INTRODUCTION

The IEEE 754 Floating Point Standard [2] will be replaced by the Type III unum posit format, which was suggested in [1]. The regime and mantissa fields of the posit format have various bit-widths and no set size. Together with the exponent bits, these regime bits create an effective exponent that increases the dynamic range of posits relative to floats. In contrast to bigger numbers, when precision is sacrificed for wider dynamic range, the posit format exhibits a crucial feature known as tapering precision, which allows smaller numbers to be represented more correctly. Once more, this results from the variable bit-widths of the posit format, which let it can express greater values than floats and to more correctly represent tiny numbers. More regime bits are used by bigger numbers than by smaller ones, resulting in fewer mantissa bits for more accurate representation. The varied bit-widths also present extra complexity for hardware design because significant machinery is needed to decode each field of the data, even if the tapering accuracy is advantageous in the majority of applications. As seen in [3], [4], and [5], a posit arithmetic block is hence more expensive than its floating-point counterpart. Interest in creating specialized hardware units for posits has decreased as a result of this disadvantage.

It is well known that deep learning applications may tolerate reduced numerical accuracy. In inferencing neural networks, high-precision calculations are not particularly helpful because they are slower and less power-efficient while also offering no benefits. According to [6], regularization effects in these neural networks can counteract the impacts of reduced accuracy. The data shown in [7], when approximation methods like truncation were used, introducing inaccuracy into the multiplier, supports this observation. Popular deep learning models such as AlexNet [8] and ResNet18 [9] used the inexact multiplier for inference calculation, and a loss of less than 1% accuracy was seen in comparison to the exact multipliers.

In comparison to the inexact multipliers in the mantissa multiplication process, the exact multipliers have an overhead of more than 2.5× and 1.5× in power and area, as demonstrated in the findings portion of this work. In addition to reducing power and space needs significantly, inexact multipliers almost eliminate accuracy loss for deep learning applications.

Background: Type I and Type II Unums

There are several variations of the unum (universal number) arithmetic framework. A "ubit" at the end of the fraction indicates whether a real number is an exact float or falls in the open interval between successive floats. The original "Type I" unum is a superset of IEEE 754 Standard floating-point format [2, 7]. The exponent and fraction field lengths fluctuate automatically, ranging from a single bit to a user-specified maximum, whereas the sign, exponent, and fraction bit fields are defined according to IEEE 754. Although type I unums offer a condensed form of interval arithmetic, additional management is required due to their changing length. By using an explicit rounding function, they are able to replicate

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

IEEE float behavior. A clean, mathematical design based on the projective reals is made possible by the "Type II" unum [4], which forgoes compatibility with IEEE floats. The most important finding is that signed (two's complement) integers map to the projective reals in an elegant manner, using the same ordering and wraparound of positive to negative quantities. They usually save store space since you're working with pointers to the values rather than actual numbers, according to William Kahan [5]. As a result, basic arithmetic may be completed quite quickly.

Figure 1 depicts the structure for 5-bit Type II unums. The "u-lattice" fills the top right quadrant of the circle with an ordered set of 2n−3 − 1 real numbers xi (which aren't always rational), with n bits per unum. The negatives of the xi, a reflection about the vertical axis, are located in the top left quadrant. × and ÷ operations are as symmetrical as + and − since the lower half of the circle contains reciprocals of the numbers in the top half, which is a reflection about the horizontal axis. The open gap between neighboring precise points, for which the unums finish in 0, is represented by Type II unums ending in 1 (the ubit), just as Type I unums. Although type II unums rely on table look-up for the majority of operations, they have numerous excellent mathematical characteristics. Although symmetries and other techniques often reduce that to a more manageable number, in the worst case, there are 22n table entries for 2-argument functions if they have n bits of accuracy. For present memory technology, the scalability of this ultra-fast format is limited to 20 bits or fewer due to table size. Additionally, Type II unums are far less suitable for fused operations. These limitations spurred an effort to find a format that would retain many of the benefits of Type II unums while being "hardware friendly," or calculable with the current float-like logic.

Posits and Valids

We compare two aesthetics for real-number calculations:

• Cheap, quick, and "good enough" for a predetermined set of applications, but not rigorous

• Meticulous and mathematical, even if it means more storage and execution time

Interval arithmetic has long been used to solve the second issue, while float arithmetic, which accepts rounding error, has long been used to address the first. One of the reasons Type I and Type II unums are referred to as "universal numbers" is that they can do either. However, it is preferable to utilize the last bit as another significant fraction bit rather than as the ubit if we are going to round after every operation using the "guess" function. We refer to an unum of this kind as a posit.

One of the restrictions is relaxed in a hardware-friendly version of Type II unums: The perfect reflection rule is exclusively followed by reciprocals for integer powers of two and 0±∞. As a result, we may fill the u lattice with finite numbers that are all of the type m· 2k, where k and m are integers, maintaining their float-like properties. Intervals are not open. Two equal-size propositions that both finish in a ubit are considered legitimate. Applications that require the rigor of interval-type constraints, such debugging a numerical method, are meant to employ them. Valids are less susceptible to quickly growing, too pessimistic boundaries and are more potent than conventional interval arithmetic [2, 4].

The Posit Format

One of the restrictions is relaxed in a hardware-friendly version of Type II unums:

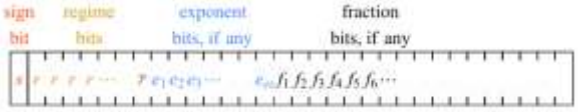

Figure. Generic posit format for finite, nonzero values

The perfect reflection rule is exclusively followed by reciprocals for integer powers of two and 0±∞. As a result, we may fill the u lattice with finite numbers that are all of the type m· 2k, where k and m are integers, maintaining their float-like properties. Intervals are not open. Two equal-size propositions that both finish in a ubit are considered legitimate. Applications that require the rigor of interval-type constraints, such debugging a numerical method, are meant to employ them. Valids are less susceptible to quickly growing, too pessimistic boundaries and are more potent than conventional interval arithmetic [2, 4]. The identical bits r are colored amber, and the opposing bit ¯ r that ends the run, if any, is colored brown. Given m, the number of identical bits in the run, k = −m if the bits are 0, and k = m−1 if its bits are 1. Decoding logic for regime bits is easily accessible because the majority of processors can "find first 1" or "find first 0" in hardware. The scale factor of used k, where used = 22es, is shown by the regime. Examples of used values are displayed in Table 2.

Binary	0000	0001	001x	01xx	10xx	110x	1110	1111
Numerical meaning, k	−4	−3	−2	−1	0	1	2	3

Run-length meaning k of the regime bits

es	0	1	2	3	4
useed	2	$2^2 = 4$	$4^2 = 16$	$16^2 = 256$	$256^2 = 65536$

The useed as a function of es

The exponent e, which is considered an unsigned integer, is represented by the following bits, which are color-coded blue. They represent scaling by 2e, therefore there is no bias like there is with floats. Depending on how many bits are left to the regime's right, there may be as many as es exponent bits. This is a concise method of conveying tapering precision; extremely big or extremely tiny numbers, which are

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

far less frequently used in computations, have less accuracy than values close to 1 in magnitude. Similar to the fraction 1.f in a float, if any bits are left over after the regime and exponent bits, they represent the fraction f, with a hidden bit that is always 1. In contrast to floats, subnormal numbers do not have a hidden bit of 0. Filling the u-lattice naturally results in the mechanism just stated. Begin with a basic 3-bit posit; just the right half of the projective reals are displayed in fig. 3 for clarity. The bit string meanings of the two extreme exception values—0 (all 0 bits) and ±∞ (1 followed by all 0 bits)—do not adhere to positional notation. As previously mentioned, the bits are color-coded for the remaining positions in Figure 3. It should be noted that in Figure 3, positive values are utilized precisely to the power of the k value that the regime represents.

By adding bits, positivity is increased, and when a 0 bit is added, values stay on the circle. A new value is created between two possibilities on the circle when a 1 bit is added. Which value ought to be attributed to each intermediate value? Let minpos be the smallest positive value on the ring formed by a bit string, and maxpos be the biggest. Minpos is 1/useed and maxpos is used in fig. The following are the interpolation rules:

• The new value is minpos/useed (new regime bit) between 0 and minpos, and maxpos×useed between maxpos and ±∞.

• The new value is their geometric mean, √ x·y =2(m+n)/2 (new exponent bit), between the current values x = 2m and y = 2n where m and n differ by more than 1.

• If not, the new value reflects the arithmetic mean (x+y)/2(new fraction bit), which is halfway between the current x and y values adjacent to it.

PROPOSED METHOD

The proposed approximate multiplier include the design of

• A novel energy-efficient posit decoder architecture with intermediate control signals to reduce dynamic power dissipation.

• A hybrid-radix Booth encoded partial product array structure for efficient mantissa multiplication.

• A high-speed mask generation circuit for variable precision depending on mantissa size of posit.

• A Booth encoding control scheme in the partial product generation (PPG) to reduce dynamic power dissipation.

2. LITERATURE SURVEY

A. POSIT FORMAT

The sign, regime, exponent, and mantissa bits make up a posit number. The changeable bit-widths of posits are the primary distinction between the IEEE 754 floating point standard and the posit standard.

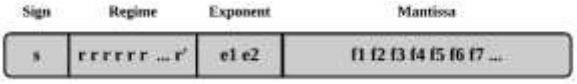

FIGURE 1. Representation of a general < N, 2 > posit.

Fig. 1 displays a generic n-bit posit representation. At the MSB, the sign bit is a single bit that is set to 1 in the case of a negative value and to 0 in the case of a positive one. A series of identical bits that are ended by their inverted bit make up the regime field. The equivalent regime value (rg) is represented by "-r" when the regime vector is made up of "r" consecutive zeros. On the other hand, the regime value (rg) is denoted as "r-1" when the regime vector displays "r" consecutive ones.

In accordance with the New Posit Standard of 2022, the exponent field's "es" number of bits has been set at two [10]. If "rg" is the regime value, "exp" is the exponent value, and "es" is the number of bits in the exponent field, then "rg × 2es+exp" is the effective exponent that is created by combining the regime and the exponent field. The remaining bits of the n-bit posit are filled by the fraction field.

The following equation yields the posit value.

$$value = (-1)^s * useed^{rg} * 2^{exp} * (1 + frac)$$

where useed = 22es

Reducing the hardware cost of basic arithmetic units to make them more energy and space efficient has received a lot of attention in the literature that is currently available. Since the mantissa or fraction multiplier is frequently the most costly in terms of both space and power, optimizing this mechanism has been given top priority in the majority of current designs. The extant literature has been divided into two categories: accurate multipliers and inexact multipliers. In order to minimize dynamic power dissipation, the precise posit multiplier suggested in [11] uses an improved leading one detector circuit in the decoder stage along with the same mantissa multiplication procedure as a conventional posit multiplier. utilizing four vertical and four horizontal control signals, the exact posit multiplier (PEfPM) created in [12] breaks down the mantissa multiplier circuit into sixteen smaller areas utilizing control logic in the PPG stage. In an effort to reduce dynamic power consumption in the circuit, the control logic only activated the parts of the circuit needed for computing based on the multiplier and multiplicand regime sizes. Compared to the multiplier in [12], the EFPM P1 and P2 multipliers that were suggested in [13] were more accurate and had better control logic schemes.

The first to investigate truncation as an approximation method in posit multiplication was the iterative shift-based posit multiplier (IPM) multiplier proposed in [14]. To conserve space, the suggested 32-bit multiplier used 12 mantissa bits and discarded the

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

remaining bits. Mitchell's Algorithm [16] served as the basis for the approximate logarithmic multiplier (PLAM) design in [15]. Despite its hardware efficiency, the PLAM architecture included a significant amount of inaccuracy. Using a radix-8 Booth algorithm for approximation multiplication, the Posit-VAR [7] has a special variable precision control module that adjusts according to the product posit's regime size, increasing precision as necessary. It also includes unconditional truncation of lower-importance partial product bits. The paper's findings shown that inexact designs provide notable benefits in terms of area, power, and time, with relatively little sacrifice in the precision of inference computation for popular deep learning models. This insight served as the impetus for the proposed study and prompted more investigation toward approximating the posit multiplication process. Table 1 highlights the characteristics of the current exact and inexact posit multipliers found in the literature.

"Beating floating point at its own game," by J. L. Gustafson and I. T. Yonemoto: Posit arithmetic,

IEEE Standard 754 floating-point numbers (floats) are intended to be directly replaced with a new data type known as a posit. In contrast to previous versions of universal number (unum) arithmetic, posits do not require variable size operands or interval arithmetic; they round, like floats, if an answer is not accurate. They do, however, have strong advantages over floats, such as broader dynamic range, better closure, higher precision, bitwise identical outputs across platforms, simpler hardware, and easier handling of exceptions. "Not a-Number" (NaN) denotes an action rather than a bit pattern, and positives never overflow to infinity or underflow to zero. Compared to an IEEE float FPU, a posit processing unit requires fewer hardware. Using comparable hardware resources, a chip's posit operations per second (POPS) can be substantially greater than the FLOPS due to its smaller silicon footprint and reduced power consumption. Particularly, GPU accelerators and Deep Learning processors can produce better results while using less power and money. Floats and posits are compared for decimals of accuracy generated for a certain precision in a thorough set of benchmarks. Compared to "approximate computing" techniques that aim to accept lower response quality, low precision posits offer a superior solution. Compared to floats of the same size, high precision posits yield more accurate decimals; in some situations, a 32-bit posit may safely replace a 64-bit float. Put differently, posits outperform floating at their own game.

Redline, Standard IEEE Floating-Point Arithmetic Standard

In computer programming environments, these standard outlines interchange and arithmetic formats and procedures for binary and decimal floating-point arithmetic. Exception circumstances and their default handling are outlined in this standard. A floating-point system that complies with this standard can be implemented fully in hardware, fully in software, or in any mix of hardware and software. The values of the input data, the order of operations, and the destination formats—all of which are controlled by the user—determine numerical outcomes and exceptions for operations listed in the normative section of this standard. Formats and operations for floating-point arithmetic in computer systems are specified by this standard. There are definitions for exception circumstances and instructions on how to handle them. This standard offers a way to compute with floating-point values that will produce the same outcome whether the processing is carried out in software, hardware, or both. With the same input data, the computation will yield the same outputs regardless of implementation. Regardless of implementation, mathematical processing errors and error situations will be communicated consistently.

Hardware implementation of POSITs and their use in FPGAs by A. Podobas and S. Matsuoka

The recent advancements in Deep Neural Networks (DNN), the failure of Dennard's scaling, and the end of Moore's law are forcing computer scientists, practitioners, and vendors to radically rethink computer design in search of possible performance gains. Reevaluating floating-point operations, which have stayed (mostly) constant over the previous thirty years, is one such option. In comparison to the IEEE-754, the POSIT numerical format, which was launched in the middle of 2017, is said to be more accurate, have a larger dynamical range, and have less wasted states. It is uncertain, meanwhile, how switching to a POSIT format may affect hardware. Our findings on POSITs and their implementation on Field-Programmable Gate-Arrays (FPGAs) are presented in this work. We created a program that automatically creates and pipelines POSIT operators, which can be used in High-Level Synthesis tools (like the Intel FPGA SDK for OpenCL) or as a drop-in replacement in processing units. We demonstrate that our units can function at similar frequencies by empirically measuring the performance and area overhead of our POSIT implementation in comparison to two well-known IEEE-754 implementations. Finally, we demonstrate how to include our POSIT hardware into the current Intel FPGA SDK for OpenCL data-paths, making it simple for software developers to carry out POSIT evaluation.

Y. Uguen, L. Forget, and F. de Dinechin, Assessing the Posit Number System's Hardware Cost

IEEE floating-point numbers are to be replaced by the posit number system. Depending on the size of the integers, this floating-point system exchanges exponent bits for significand bits. As a result, it offers more accuracy for numbers close to one at the price of less accuracy for extremely big or extremely small

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

integers. Numerous studies have shown that this trade-off can increase an application's accuracy. However, the hardware cost of posit arithmetic is affected by the variable-length exponent and significand encoding. Enabling application-level assessments of the overall system that take performance and resource use into account is the aim of the current effort. This article presents a C++ templated library that is compatible with Vivado HLS, which is an open-source hardware implementation of the posit number system. For custom-size propositions, this library presently provides addition, subtraction, and multiplication. The inclusion of the "quire," a large accumulator capable of performing precise product sums, is also required by the posit standard. The first open-source parameterized hardware quire is included in the suggested library. It has been demonstrated that this library enhances the state-of-the-art in existing implementations with respect to resource utilization and latency. However, compared to the analogous floating-point operators, typical 32-bit posit adders and multipliers are shown to be significantly slower and bigger. It is demonstrated that the price of the posit 32 quire is similar to that of a Kulisch accumulator for 32-bit floating-point.

Effective posit multiply-accumulate unit generator for deep learning applications by H. Zhang, J. He, and S.-B. Ko

The new posit number system is more precise than the traditional IEEE754-2008 floating-point numbers and can offer a larger dynamic range. It is appropriate for deep learning applications due to its nonuniform data representation. Recent years have seen the development of the posit adder and posit multiplier in the literature. Nevertheless, no research has yet been done on the use of posit in fused arithmetic units. This work proposes an efficient design of posit multiply-accumulate (MAC) unit to enable the usage of posit number format in deep learning applications. The posit format is more adaptable than IEEE754-2008, which has four standard binary number formats. The total bitwidth and exponent bitwidth can be any value. Thus, a custom MAC unit generator written in C is suggested in this architecture, along with parameterizing the bitwidths of every datapath. For every specified total bitwidth and exponent bitwidth, the suggested generator may produce Verilog HDL code of the corresponding MAC unit. Although the code produced by the generator is a combinational design, this study also presents and analyzes a 5-stage pipeline technique. With various bitwidth selections, the worst-case latency, area, and power consumption of the created MAC unit under the STM-28nm library are given and examined.

H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, D. Kudithipudi, Z. Carmichael, Low-precision numerical representations' performance-efficiency trade-off in deep neural networks

Deep neural networks (DNNs) have shown to be successful prognostic models in a number of fields, including genomics, computer vision, and natural language processing. However, in order to do any decently difficult job, contemporary DNNs require a large amount of CPU and memory storage. DNN accelerators using low-precision representations of input and DNN parameters are being intensively researched in order to reduce the power consumption of these networks and maximize the inference time. How to move low-precision networks to edge devices with comparable performance to high-precision networks is an intriguing research subject. In this study, we use exact multiply-and-accumulate (EMAC) units for inference and the fixed-point, floating point, and posit numerical representations at <8-bit precision within a DNN accelerator, Deep Positron. The trade-offs between overall network efficiency and performance across five categorization tasks are quantified in a single analysis. In comparison to floating point, our results show that posits are a natural match for DNN inference, outperforming at <8-bit precision, and can be implemented with competitive resource needs.

S.-B. Ko and H. Zhang, Effective approximation posit multipliers for computing in deep learning

In recent years, there has been an increasing interest in the format of positive numbers. It is particularly well-suited for a wide range of applications, including deep learning computing, because to its tapering accuracy. However, posit arithmetic is more expensive to implement in hardware than its floating-point version because of its changeable component bit-width. This study proposes posit multipliers with approximate computing properties to address this cost issue. The fundamental concept of the suggested design is to truncate the fraction multiplier in accordance with the product's estimated fraction bit-width. in order to greatly lower the resource usage of the fraction multiplier and, consequently, the fraction adder. The suggested approach is used for constructing multipliers in both the linear and logarithmic domains. The suggested approximation posit multipliers are implemented and examined in their 8/16/32-bit variant. The suggested approximation posit multiplier can utilize 16% less power than the traditional posit multiplier design for the widely used 16-bit posit format in deep learning computing. When compared to the state-of-the-art existing approximate logarithm multiplier, the suggested 16-bit approximate logarithm multiplier can achieve a 15% improvement in power consumption. Numerous deep neural network models are computed using the suggested 16-bit approximation posit

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

multipliers, which provide in notable energy efficiency gains with minimal accuracy compromise.

Deep convolutional neural networks for ImageNet classification by A. Krizhevsky, I. Sutskever, and G. E. Hinton

To categorize the 1.2 million high-resolution photos in the ImageNet LSVRC-2010 competition into the 1000 distinct classes, we trained a large, deep convolutional neural network. We significantly outperformed the prior state-of-the-art with top-1 and top-5 error rates of 37.5% and 17.0% on the test data, respectively. With 60 million parameters and 650,000 neurons, the neural network is made up of three fully-connected layers, a final 1000-way softmax, and five convolutional layers, some of which are followed by max-pooling layers. We employed non-saturating neurons and a very effective GPU version of the convolution process to speed up training. We used a recently created regularization technique called "dropout" to lessen overfitting in the fully-connected layers, and it worked really well. In the ILSVRC-2012 competition, we also entered a variation of this model and won with a top-5 test error rate of 15.3%, whereas the second-best entry had a top-5 error rate of 26.2%.

"Deep residual learning for image recognition," by K. He, X. Zhang, S. Ren, and J. Sun, 2015,

Training deeper neural networks is more challenging. In order to facilitate the training of networks that are far deeper than those previously employed, we provide a residual learning approach. Rather than learning unreferenced functions, we explicitly reformulate the layers as learning residual functions with reference to the layer inputs. We offer thorough empirical proof that these residual networks are simpler to tune and that a significant increase in depth can improve accuracy. We test residual nets with a depth of up to 152 layers on the ImageNet dataset, which is 8× deeper than VGG nets [40] but still less complicated. On the ImageNet test set, an ensemble of these residual nets achieves an error of 3.57%. On the ILSVRC 2015 classification task, this result took first place. Analysis of CIFAR-10 with 100 and 1000 layers is also presented. For many visual identification tasks, the depth of representations is crucial. We achieve a 28% relative improvement on the COCO object detection dataset, purely because of our very deep representations. Our contributions to the ILSVRC & COCO 2015 competitions1 were built on deep residual nets, and we took first place in the ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation tasks.

R. S. Haripriya, Low-delay, energy-efficient posit multiplier design

The hardware implementation of the Posit Multiplier unit is the main topic of this study. Since it yields more accurate results over the same word size, the Posit number system is an alternative to the IEEE-754 Floating Point Unit. The run-time-changing exponent component, which combines regime bits (with run-time varying duration) and exponent bits, improves the dynamic range. The multiplexer-based logic used by the current leading one detector accounts for additional area and data channel delay in the higher-order posit arithmetic units. To cut down on area and latency, a Leading One Detector based on an adder has been suggested. For each given posit data width (N), exponent size (ES), and regime size (RS), the suggested Posit multiplier extracts the data bits from the posit data using the suggested LOD, resulting in reduced area and data route latency. Because the adder size is reduced, the area is further improved by the independent computation of the final exponent and final regime. We created the 8-, 16-, and 32-bit posit multiplier units and contrasted the outcomes with previous research. According to the synthesis results, there was a 17.55% decrease in delay and a 28.35% gain in energy efficiency.

3. PROPOSED METHODOLOGY

3.1 Methodology

Specifically for the < 16, 2 > posit format, the proposed work entails designing an approximation posit multiplier (ADEPNET) circuit. Together with the particular arithmetic operation, any mathematical operation on a posit requires both encoding and decoding. In the case of posit multiplication, posit decoding and encoding are carried out in addition to the conventional mantissa multiplication, effective exponent addition, and sign computation. Floating-point operations lack these decoding and encoding circuits, which are intrinsic to conventional arithmetic. of contrast to a floating-point number, where each field takes up a constant number of bits, these additional steps are caused by the varied bit-widths of the various fields of a posit. Fig. 2 displays the whole posit multiplication data flow, illustrating each step from initial decoding to final encoding. The mantissa multiplication circuit and the suggested posit decoder circuit are indicated in grey. An < n, es > posit can be broken down into its component parts—sign, regime, exponent, and mantissa—using a posit decoder circuit. Rg × 2es+exp is the formula for the effective exponent, which is formed by the regime and the exponent field. Here, rg is the regime value, exp is the exponent value, and es is the number of bits in the exponent field. The resulting effective exponent is obtained by adding the effective exponents of the two propositions after the effective exponent has been determined. The resulting sign bit is obtained by XORing the various sign bits, and the resultant mantissa is formed by multiplying the mantissas. After normalizing the mantissa and exponent, the encoder packs the resulting mantissa, effective exponent, and sign bits to create the resultant posit.

A. POSIT DECODER PROPOSED

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

One essential requirement for carrying out any posit-based arithmetic operation is the posit decoder seen in Figure 3. For integer arithmetic circuits to approach or surpass their floating-point equivalents in performance metrics, decoding overhead minimization becomes crucial, unlike floating-point operations that do not require this step.

When the sign bit is represented as 1, the input is subjected to a twos complement operation in common design paradigms. The effective exponent and mantissa values are then extracted through further processing that includes an additional XOR operation with the MSB of the regime field. The effective exponent is not only the exponent field's contribution; it is the overall exponent of the input data from the regime and exponent fields. The twos complement operation, which integrates intermediate control signals to account for all input possibilities, is partially processed in the suggested design framework. This method increases the overlap between the exponent and mantissa computation while decreasing the breadth of the complementing adder.

Additionally, by decreasing the multiplier's total latency, the suggested method increases its usefulness. By providing the regime value—which the mantissa multiplier needs as an input—available sooner, this is achieved. There are four scenarios in which the sign bit and first regime bit are (0,0), (0,1), (1,0), and (1,1), respectively, while decoding the regime field of posits. The regime field adds "-p" to the effective exponent in situations (0,0) and (1,1), where "p" is the number of times the first regime bit repeats before being terminated by the opposite bit. Alternatively, it adds a factor of "p - 1" in situations (0,1) and (1,0). An additional adder is not required because the control signal "ctrl" is activated upon detection of (0,1) or (1,0) situations for hardware efficiency. When the "ctrl" bit is set, the sign bit is added to the end of the changed input after the input to the leading bit counter has been moved one bit to the left, decrementing the count by one. When the remainder of the input is all ones or zeros, the sign bit can serve as the terminating bit because it is always different from the first regime bit in these situations. The sign bit and the most important regime field bit are XORed to create an intermediate control signal, or "ctrl," which permits the input to be processed in a different order. The multiplexer uses the intermediate control signal, which is produced from the sign bit and first regime bit, as a select input to decide which signal should be sent to the leading bit counter. The left-shifted posit is used as the input when the sign bit and the first regime bit vary. On the other hand, the unaltered posit bits are chosen when the two bits are the same. Furthermore, the intermediate control signal's importance goes beyond the multiplexer; it may also be used as an adder's input. The leading bit counter output and the intermediate control signal are used as the adder's inputs. Together, they establish the exact shift amount that is necessary to correctly decode the input position's exponent and mantissa fields. Following the shifting operation, the sign bit determines how the exponent and mantissa fields are extracted and handled. The pre-processed mantissa bits undergo NOR reduction to create a new signal called "sadd_cin," which is then used as the carry-in to the adder in the exponent processing stage. The sign of the effective exponent is represented by the signal "ss."

B. MANTISSA MULTIPLIER PROPOSED

Since the maximum mantissa size of a < 16, 2 > posit is 12, the mantissa multiplier circuit has been developed for a 12-bit operand size. The < 8, 0 > posit has also been approached using the similar manner. Since the mantissa multiplication step adds to the circuit's maximum size, power, and delay, there has previously been a lot of focus on lowering its hardware cost. Several approximation methods and power optimizations that may be used in the mantissa multiplication process are examined in this research. The following stages, each with a distinct contribution, comprise the overall Mantissa multiplication process:

1. Partial Product Array Encoded using Hybrid-Radix Booth
2. Module for Mask Generation
3. Control of Booth Encoding

1) Encoded Partial Product Array For Hybrid-Radix Booth

The Booth multiplication method is used in the Mantissa multiplication process, however it does not employ a single radix encoding approach as other posit multipliers do. In order to circumvent the inherent restrictions of the radix-4 and radix-8 schemes and integrate their respective benefits, a hybrid Booth encoding has been developed.

Because there are less partial product rows in the radix-8 encoding than in the radix-4 encoding, the reduction of partial product rows is easier. But it also requires calculating three times the multiplicand, which is only possible by addition. The delay caused by addition in radix-8 encoding, which is absent in radix-4 encoding, cancels out the benefit of having fewer partial products. The impact of addition might be lessened by approximating the addition operation in a radix-8. But the system also experiences serious mistakes as a result of this. As a result, a hybrid Booth encoding technique has been developed in place of a strictly radix-8 encoding, where the following two partial product rows are encoded using an exact radix-4 scheme, while the first three partial product rows are encoded using radix-8. In contrast to a solely radix-8 architecture, the hybrid encoding technique depicted

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

in Fig. 4 guarantees that the more significant partial product rows are precisely computed to lower the system's error without increasing the number of partial products.

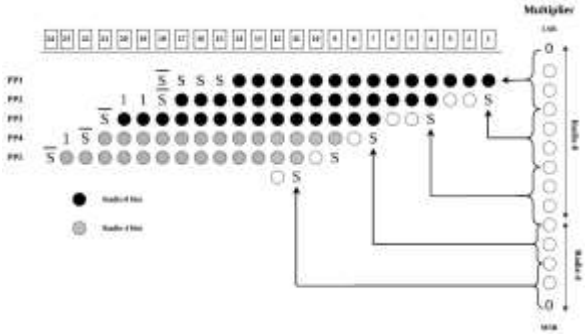

FIGURE 4. Hybrid-radix encoded partial product array

The sign extension of each partial product row in Fig. 4 is made up of the extended sign bits and "1" bits seen in that row. In the conventional Booth process, all partial products are sign extended so that the last bit position of each partial product row corresponds to a column number of "2n," where the multiplicand is n bits wide. Reduced sign extensions for various partial product rows have been created for both radix-4 and radix-8 Booth multipliers since this extension is not area-efficient, as seen in Fig. 4.

Three times the multiplicand (3M) must be calculated for the radix-8 scheme. This is more difficult than calculating twice the multiplicand (2M), which is readily done with a left shift operation. Therefore, instead of calculating the precise total, the 3M partial product computation has been estimated. As seen in Fig. 5, a 7-bit precise addition on the MSB side of M and 2M has been carried out using a carry-in prediction circuit in place of a 12-bit addition between M and 2M. The approximation significantly reduces the circuit's total power, latency, and area.

The least significant bit in the partial product array that is not truncated is the P6 bit in the 3M partial product computation. The bits utilized to calculate the total are multiplicand bits A11–A5. Because the carry-in calculation has not been completed, there is an error in the 3M partial product calculation. To get around this, a carry-in prediction circuit that uses the multiplicand's A5, A4, and A3 bits to anticipate the carry-in to the 7-bit addition has been proposed. The carry-in prediction circuit predicts a logic one if any combination of the following pairs—A5, A4, and A3—are both at a logic high.

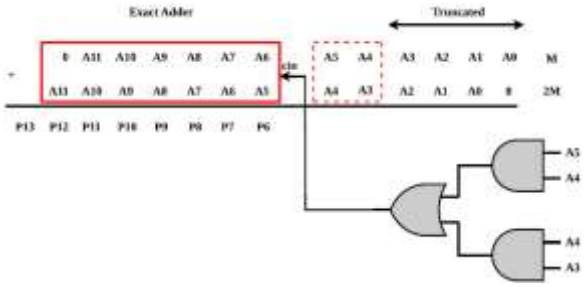

FIGURE 5. Approximation for 3M partial product calculation with carry prediction scheme

2) MASK GENERATION CIRCUIT

The maximum number of bits in the mantissa field, including the inferred bit, for a < 16, 2 > posit is 12. The size of the individual mantissa operands is 12 apiece, therefore when the mantissa multiplication operation is carried out, the mantissa of the result is 24 bits. The full multiplication procedure is superfluous since the mantissa size is restricted to 12 bits. Instead, partial product bits can be reduced such that just the partial product bits that correspond to the 13 most important columns are produced. As seen in Fig. 6, 13 columns of the partial product array have been created for normalization and rounding reasons, despite the fact that the posit mantissa size is restricted to 12 bits.

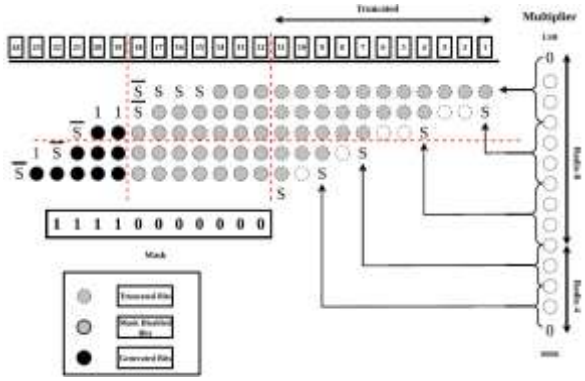

FIGURE 6. Proposed mantissa multiplier with bit masking.

There are further optimizations to lower dynamic power in the case of posits besides unconditional truncation of the columns on the LSB side. As the size of the regime field increases in a pose, the mantissa field's size decreases. Consequently, the mantissa field's size is at its largest when the regime size is at its smallest. Only when the maximum size of the mantissa is used is the unconditional truncation taken into consideration. Additional columns of the partial product array can be masked in addition to the current truncation, as seen in Fig. 6, when the size of the product mantissa's mantissa field is less.

Depending on the ultimate regime size of the product posit, this results in a dynamic precision with different amounts of approximation. This is achieved by employing a mask generating circuit, which,

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

depending on the width of the regime field, generates a signal to deactivate the partial product bits during the partial product production step. The regime width in a < 16, 2 > posit can range from 2 to 15 bits. The width of the regime field is the only factor that determines the amount of mantissa bits because the exponent size is fixed at two bits. Consider a scenario in which the number of mantissa bits in the posit is 10 (not including the inferred bit), the regime value is 1, and the regime field width is 3. The least important mantissa bit in this situation can be hidden to conserve power because the maximum mantissa size is 11. Table 2 illustrates the relationship between the regime field value, regime field width, mantissa field width, and matching mask value.

Reg_Val [3:0]		Reg Width	Mant Width	Mask String
Reg_Val[4] = 0	Reg_Val[4] =1			
0000	1111	2	11	11111111111
0001	1110	3	10	11111111110
0010	1101	4	9	11111111100
0011	1100	5	8	11111111000
0100	1011	6	7	11111110000
0101	1010	7	6	11111100000
0110	1001	8	5	11111000000
0111	1000	9	4	11110000000
1000	0111	10	3	11100000000
1001	0110	11	2	11000000000
1010	0101	12	1	10000000000
1011	0100	13	0	00000000000
1100	0011	14	0	00000000000
1101	0010	15	0	00000000000
1110	0001	15	0	00000000000
1111	0000	15	0	00000000000

TABLE 2. Mask signal generation truth table.

To lessen dynamic power dissipation, the produced mask values are transmitted to the partial product array as column-wise enable signals. The partial product array in Figure 6 combines mask-based enabling and disabling of other columns according to the size of the regime field with unconditional truncation of the columns of lower columns. The regime value of the product in the case shown in Figure 6 is 7, which translates to a regime field size of 9 bits and a mantissa size of 4 bits (not including the implied bit). In this instance, the mask inhibits the 7-bits near the LSB side and activates the 4-bits of the mantissa product on the MSB side. Since it matches the mantissa's inferred bit, the most important column is always left uncovered. Table 2 makes it clear that a straightforward left shift operation on a string of "11" ones can provide the mask, with the magnitude of the shift depending on the regime value of the product postulate. The shift amount for the left shifter circuit is set to the four bits on the LSB side of the regime value field if the regime value is positive. The one's complement of the four bits on the LSB side of the regime value is used as the shift amount when the regime value is negative. The mask has been generated using an 11-bit barrel shifter circuit. The circuit suggested in [7], which was created using Boolean logic equations obtained from the truth table, is different from the mask creation. The suggested circuit makes use of a straightforward architecture based on a left shifter that is readily scaled to fit any general size position. While the space and power of the suggested mask generating circuit are similar to those of the circuit in [7], the latency is improved.

3) BOOTH ENCODING CONTROL

In order to minimize dynamic power dissipation, partial product bits have been enabled and disabled using the mask circuit after they have been produced. By sending enable signals from the mask generation circuit to the Booth encoding itself, it is possible to further lower the dynamic power. In certain instances, the mask turns off every partial product bit in a certain partial product row; as a result, these bits are not used in the product computation. It is a waste of power to generate these bits from the Booth encoding and PPG blocks because they will always be masked. Even more power can be saved by constraining the PPG blocks' output to a logic low if the Booth encoding for the full row of partial products can be turned off. For instance, the mask circuit disables columns 12, 13, and 14 if the mask value is "111111111000." There are no partial product bits in the row that are enabled by the mask since the MSB of the first partial product row is located in column 14. All of the partial product bits are set to a logic low if the Booth encoding, which is typical of a certain row of partial products, is turned off. The MSB of the row in column 12 is necessary for the product calculation and cannot be set to a logic low, hence the Booth encoding for the first partial product row cannot be disabled if the mask value is "11111111110." Thus, the Booth encoding may be turned off for specific mask values alone.

The Booth encoding circuits of rows 1, 2, 3, 4, and 5 have been enabled by the mask bits that correspond to columns 14, 17, 20, 21, and 23 accordingly. The enable signal for the Booth encoding for a given row has been the mask bit that corresponds to the column that contains the MSB of that row. Since the mask is a thermometric string, this has been done. If a certain bit is at a logic 0, then all the bits to its right are likewise at a logic 0. Thus, the mask bits corresponding to every other bit in a given row will likewise be 0 if the mask bit corresponding to the MSB of that row is 0. In this instance, it is feasible to optimize the partial product row by turning off Booth encoding.

Fig. 7 displays a Booth encoder circuit with a PPG block after it. The encoding is chosen based on the multiplier bits "x[2:0]". The partial product bit is set to a logic low and cannot toggle if all three signals—one, two, and neg—are at a logic low. This lowers the dynamic power since the partial product bit is deactivated at the encoding stage itself rather than being generated and subsequently masked. It should be noted that even if a whole row of partial product

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

bits is disabled, the extended sign bits of that specific partial product row are still present in the partial product array. The input bit "x[2]" to the Booth encoder circuit, as illustrated in Fig. 7, is used as the sign bit rather than the "neg" signal set to 0. This guarantees that, regardless of the Booth encoding control, the sign bits remain a part of the partial result.

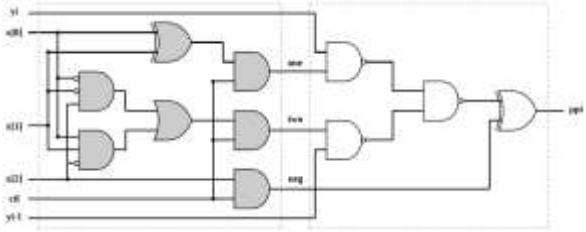

FIGURE 7. Booth encoding control.

5. SIMULATION RESULTS:

Verilog HDL has been used to provide a fair comparison between the suggested and current designs. At a TSMC 90 nm manufacturing node, the Cadence RTL compiler v7.1 was used to create the area, power, and delay reports of the proposed ADEPNET, the current designs PEfPM [12], Posit [1], and Posit-VAR [7]. The slow-normal library has been used particularly for the synthesis, which was carried out at 1.8V, 25 C, and 1 GHz.

Table 3 presents the results of the proposed study as well as a thorough analysis of precise and approximate mantissa multipliers. Since the mantissa multiplication operation is the most resource-intensive in a posit multiplier, Table 3 only shows the mantissa multiplication results. This highlights how, in comparison to their precise counterparts, the approximate multipliers appear to have used less hardware resources. The suggested ADEPNET is an approximate posit multiplier, while the other 16-bit posit multiplier designs, PEfPM [12], Posit [1], and Posit-VAR [7], are accurate posit multipliers. It is clear that the inexact mantissa multiplier of Posit-VAR [7] provides a minimum decrease in area, power, and latency of 39%, 71%, and 26%, respectively, in comparison to the exact mantissa multipliers. The suggested ADEPNET architecture further expands these power, area, and delay advantages. The ADEPNET and Posit-VAR both make use of truncation, which results in a partial product array that is about half as large as the current precise designs. This helps to significantly reduce the area. Depending on the regime size of the device, the unconditional truncation and masking of superfluous partial product

bits lower the circuit's partial product bits' switching activity, which significantly lowers dynamic power consumption.

FIGURE: DEVICE UTILIZATION REPORT

FIGURE: POWER UTILIZATION REPORT

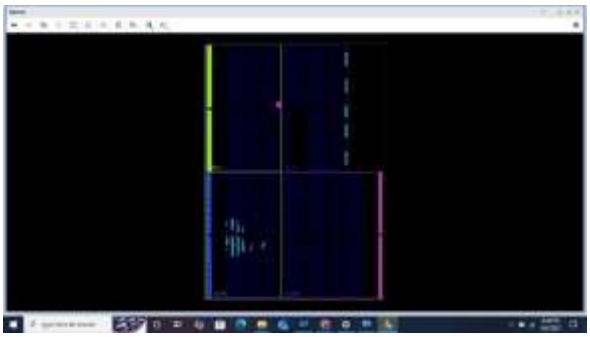

FIGURE: POST IMPLEMENTATION

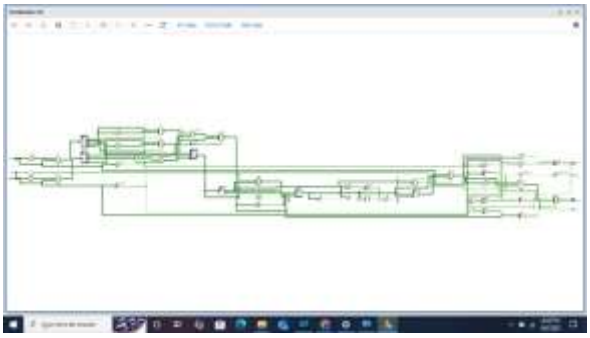

FIGURE: RTL SCHEMATIC

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

FIGURE: SIMULATION OUTPUT

6. CONCLUSION

For deep learning applications, this work investigates approximation posit multipliers as a substitute for exact posit multipliers and floating-points. In order to lower hardware resource consumption, an approximate multiplier circuit has been developed using the higher precision positions that are offered. The suggested approximate design shows a power dissipation of less than 23% and a significant 46% reduction in area when compared to the exact PEfPM design in the mantissa multiplication stage by employing approximations such as truncation, bit masking, and the introduction of a novel Booth encoding control scheme. As demonstrated in the results section, where the approximate design had less than 1% accuracy degradation in inference accuracy compared to the exact posit multipliers, the suggested approximations result in a significant reduction in hardware complexity and have no discernible impact on the accuracy of inference of deep learning models. Future research will investigate a combined usage of < 16, 2 > and < 8, 0 > posits in an effort to improve hardware efficiency of inference computation without appreciably lowering accuracy.

REFERENCES

[1] J. L. Gustafson and I. T. Yonemoto, ‘‘Beating floating point at its own game: Posit arithmetic,’’ *Supercomput. Frontiers Innov.*, vol. 4, no. 2, pp. 71–86, 2017.

[2] *IEEE Standard for Floating-Point Arithmetic—Redline*, Standard IEEE Std 754-2008, Aug. 2008, pp. 1–82, doi: 10.1109/IEEESTD.2008.5976968.

[3] A. Podobas and S. Matsuoka, ‘‘Hardware implementation of POSITs and their application in FPGAs,’’ in *Proc. IPDPSW*, May 2018, pp. 138–145.

[4] Y. Uguen, L. Forget, and F. de Dinechin, ‘‘Evaluating the hardware cost of the posit number system,’’ in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 106–113.

[5] H. Zhang, J. He, and S.-B. Ko, ‘‘Efficient posit multiply-accumulate unit generator for deep learning applications,’’ in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[6] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, ‘‘Performance-efficiency trade-off of low-precision numerical formats in deep neural networks,’’ in *Proc. Conf. Next Gener. Arithmetic*, 2019, pp. 1–9.

[7] H. Zhang and S.-B. Ko, ‘‘Efficient approximate posit multipliers for deep learning computation,’’ *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 201–211, Mar. 2023, doi: 10.1109/JETCAS.2022.3231642.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ‘‘ImageNet classification with deep convolutional neural networks,’’ in *Proc. NIPS*, vol. 1, 2012, pp. 1097–1105.

[9] K. He, X. Zhang, S. Ren, and J. Sun, ‘‘Deep residual learning for image recognition,’’ 2015, *arXiv:1512.03385*.

[10] Posit Working Group, *Posit Standard Documentation, Release 3.2-Draft*, 2018. Accessed: Sep. 24, 2020. [Online]. Available: https://posithub.org/docs/posit standard.pdf

[11] R. S. Haripriya, ‘‘Design of energy efficient and low delay posit multiplier,’’ in *Proc. 36th Int. Conf. VLSI Des. 22nd Int. Conf. Embedded Syst. (VLSID)*, 2023, pp. 1–9.

[12] H. Zhang and S.-B. Ko, ‘‘Design of power efficient posit multiplier,’’ *IEEE Trans. Circuits Syst. II, Exp. Papers*, vol. 67, no. 5, pp. 861–865, May 2020, doi: 10.1109/TCSII.2020.2980531.

[13] A. A. Jonnalagadda, A. K. Uppugunduru, S. Veeramachaneni, and S. E. Ahmed, ‘‘Design of energy efficient posit multiplier,’’ in *Proc. Great Lakes Symp. VLSI*, Jun. 2023, pp. 1–6.

[14] C. J. Norris and S. Kim, ‘‘An approximate and iterative posit multiplier architecture for FPGAs,’’ in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jul. 2021, pp. 1–5.

[15] R. Murillo, A. A. Del Barrio, G. Botella, M. S. Kim, H. Kim, and N. Bagherzadeh, ‘‘PLAM: A posit logarithm-approximate multiplier,’’ *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 2079–2085, Oct. 2022, doi: 10.1109/TETC.2021.3109127.

[16] J. N. Mitchell, ‘‘Computer multiplication and division using binary logarithms,’’ *IRE Trans. Electron. Comput.*, vols. EC–11, no. 4, pp. 512–517, Aug. 1962.

[17] A. D. Booth, ‘‘A signed binary multiplication technique,’’ *Quart. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.

[18] K. Simonyan and A. Zisserman, ‘‘Very deep convolutional networks for large-scale image recognition,’’ 2014, *arXiv:1409.1556*.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, ‘‘Gradient-based learning applied to document recognition,’’ *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, ‘‘Densely connected convolutional networks,’’ in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 2261–2269, doi: 10.1109/CVPR.2017.243.

[21] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H.Wu, ‘‘Mixed precision training,’’ 2017, *arXiv:1710.03740*.