International Journal on Applications in Engineering and Technology
Volume 9: Issue 5 : 2023, pp 16 – 24 www.aetsjournal.com ISSN (Online) : 2455 - 0523

DESIGN OF POWER EFFICIENTPOSIT MULTIPLIER

MERLIN *M.E.Student Department of el Electronics and Communication Engineering, Tagore institute of Engineering and Technology, Attur 636112, MAIL ID : merlinece@gmail.com*
HARITHA *M.E, Assistant Professor, Department of el Electronics and Communication Engineering, Tagore institute of Engineering and Technology, Attur 636112Mail: haritha.ece@tagoreiet.ac.in*

Abstract**—** Posit number system has been used as an alternative to IEEE floating-point number system in many applications, especially the recent popular deep learning. Its non-uniformed number distribution fits well with the data distribution of deep learning and thus can speed up the training process of deep learning. Among all the related arithmetic operations, multiplication is one of the most frequent operations used in applications. However, due to the bit-width flexibility nature of posit numbers, the hardware multiplier is usually designed with the maximum possible mantissa bit-width. As the mantissa bit-width is not always the maximum value, such multiplier design leads to high power consumption especially when the mantissa bit-width is small. In this brief, a power efficient posits multiplier architecture is proposed. The mantissa multiplier is still designed for the maximum possible bit-width; however, the whole multiplier is divided into multiple smaller multipliers. Only the required small multipliers are enabled at run-time. Those smaller multipliers are controlled by the regime bit-width which can be used to determine the mantissa bit-width. This design technique is applied to 8-bit, 16-bit, and 32-bit posit formats in this brief and an average of 16% power reduction can be achieved with negligible area and timing overhead. designs of FAs are also proposed in this article utilizing the proposed XOR–XNOR circuit and available sum and carry modules. The proposed FAs provide 2%–28.13% improvement in terms of PDP than that of other architectures. To measure the driving capabilities, the proposed FAs are embedded in 2-, 4-, and 8-bit cascaded full adder (CFA) structures. Results show that two of the proposed FAs provide the best performance for a higher number of bits among all theFAs. The proposed XOR–XNOR module is implemented with ten transistors, and it has a symmetrical structure which makes the layout of the proposed XOR–XNOR circuit less complex.

I. INTRODUCTION

Multipliers play an important role in today's digital signal processing and various other applications. In high performance systems such as microprocessor, DSP etc. addition and multiplication of two binary numbers is fundamental and most often used arithmetic operations.

II. COMPREHENSIVECASE STUDY

Multiple floating-point representations have been used in computers over the years, although the IEEE Standard for Floating-Point Arithmetic (IEEE 754) [2] is the most common implementation that modern computing systems have adopted. Since it was established in 1985, the standard has only been revisited in 2008 (IEEE 754-2008) [3], but it remains the main characteristics of the original to keep compatibility with existing implementations and it is not adopted by all computer systems. However, multiple shortcomings have been identified in the IEEE 754 standard, which are listed below [4]: Different computers using the same IEEE floating-point format are not required produce the same results. When a computation does not fit into the chosen number representation, the number will be rounded. Even in the last revision of the standard they introduce the *round-to-nearest, ties away from zero* rounding schemes and provide recommendations for computations reproducibility, hardware designers are not coerced to implement them. Therefore, identical computations can lead to multiple results across different computing platforms [5].Multiple bit patterns are used for handling exceptions such as the Not-A-Number (NaN) value, which indicates that a value is not representable or undefined – for example dividing by zero results in a NaN. The problem is that the amount of bit patterns that represent NaNs may be more than necessary, making hardware design more complex and decreasing the available number of

exactly representable values. IEEE 754 makes use of overflow – accepting ∞ or -∞ as a substitute for large magnitude finite numbers – and underflow – accepting 0 as a substitute for small magnitude nonzero numbers. Thus, major problems can be produced, as the above mentioned. Rounding is performed on individual operands of every calculation, so associativity and distributive properties are not always held in floating-point representation. The last revision of the standard tries to solve this issue including the fused multiply–add (FMA) operation. However, again this may not be supported by all computer systems Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation. The common multiplication method is "add and shift" algorithm. In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms. To achieve speed improvements Wallace Tree algorithm can be used to reduce the number of sequential adding stages. Further by combining both Modified Booth algorithm and Wallace Tree technique we can see advantage of both algorithms in one multiplier. However with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced speed, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing. On the other hand "serial-parallel" multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this lecture we introduce the multiplication algorithms and architecture and compare them in terms of speed, area, power and combination of these metrics. Number systems and computer arithmetic's are essential for designing efficient hardware and software architecture. In particular, real-valued computation constitutes a crucial component in almost all forms of today's computing systems from mobile devices to servers. IEEE 754 is a prominent standard established in 1985 for representing real-valued numbers in a floating-point format. Despite all its benefits, this number system suffers from a number of weaknesses.Large size for small numbers. The IEEE 754 standard defines two specific formats for single- and double-precision value representation using 32 and 64 bits, respectively. Numerical computation within a limited range of values in these formats may be largely inefficient.For example, computing dot-products of values within [-1, 1] requires only a tiny fraction of all possible numbers represented in either format.

- Limited precision. IEEE 754 has predefined fixed-size partitions for an exponent and a fraction. This may lead to a rounding error when representing real numbers; therefore, some floating-point numbers are not precise.
- Exceptional bit representations. IEEE 754 reserves several bits to represent NaNs, de normals, positive/negative zero and infinity. In addition to wasting some of the possible bit patterns, considering all the reserved patterns for computation adds further complexity to the floating-point processors.
- Breaking algebraic rules. The floating-point formats may break algebraic rules during computation. For instance, a floating-point addition is not always associative. The expression (x+y)+z results in 1, where the floating-point values are x = 1e30, y = -1e30 and z = 1 is 1. Using the same values, x+(y+z) results in 0.
- Producing inconsistent results. Consider two vectors Q = (3.2e7, 1, –1, 8.0e7) and W = (4.0e7, 1, –1, –1.6e7). The dot product $Q.W$ is equal to 0 in single-precision (i.e., the float type in C); while, the right answer is 2. Using the floating-point representation, 80 intermediate bits are

necessary to produce the correct answer in the double-precision format.

- Complex design and verification. Designing an efficient floating-point unit could be time-consuming due to the necessary components for handling rounding, exception, NaNs, denormals, mantissa alignment, etc. Moreover, verifying the floating-point design is a significant task because of dealing with numerous corner cases.

To overcome these challenges, various number systems and data representation techniques have been proposed to enhance or replace the floating-point numbers. A few examples of such techniques are Interval arithmetic, universal number systems Type I, and Type II. The most recent floating-point number system invented by John L. Gustafson in 2017 is called posit that addresses many of the above-mentioned problems. Most machine implementations of real numbers rely on floating-point arithmetic. The ease-of-use of floating-point, which explains its popularity, hides complex hardware whose behavior is specified by the IEEE-754 standard. The posit number system (described in details in [2]) is an emerging machine representation of real numbers that aims at replacing IEEE-754 floating-point. The first posit claim is that floating-point is an inefficient representation. When the exponent can be encoded on only a few bits, the rest of the bits should be used to extend the precision. The second claim, adopted from Kulisch [3], is that the sum of many products is a pervasive operation, justifying specific hardware to compute it exactly. To this purpose, the draft posit standard [4] mandates a quire, a variant of the exact Kulisch accumulator [3] for the posit number system. Most current evaluations of posits in applications are performed through software simulation [5], [6], [7], [8]. The C/C++ Soft Posit library 1 (among others 2) implements the latest posit standard and allows for direct comparison with floating-point numbers in terms of accuracy. However, the hardware cost of posits is not yet completely known. Hardware posits adders and multipliers have been written in HDL [9], [10] or using Intel Open CL SDK compliant templatized C++ operators [11]. Using posits as a storage format by decoding/encoding from/to a large enough IEEE floating-point format as also been studied in [5]. Posits have been evaluated on applications such as machine learning [5],[6] or matrix multiply [7]. Among these works, only [5] is open-source and partially supports the quire, but only for 8bit posits. [11] and [9] are parametric designs but are not open-source and do not support the quire. The present work, although similar in spirit, refines the architectures in [11], attempting to use the same data path optimization tricks that are used in the floating-point operators it compares to [12]. Conversely, [9] compares a posit implementation to a floating- point implementation that is 3x larger than the state-of-the-art. The present work improves the implementation of posit hardware with respect to all the previous works, and enables a comparison with state-of-the-art floating-point. It is parametric, open-source, and it is the first implementation to include a standard-compliant, parametric quire. As the quire is the posit incarnation of the exact Kulich accumulator for IEEE floating-point, an implementation of the latter is provided for good measure.

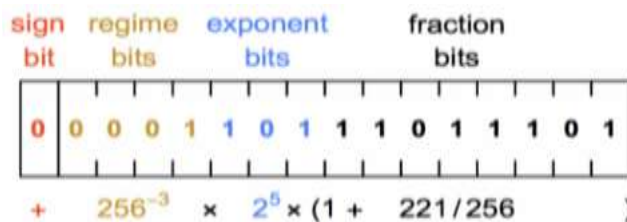

Figure 1: Example of a posit number and its decimal value

Posit Format

Fig. 1 shows values for a 3-bit posit format with $n = 3$ and $es = 1$. There are only two reserve representations: 0 (all 0 bits) and ±∞ (1 followed by all 0 bits). A total of 8 values may be represented using 3 bits. A generic posit format consists of a mandatory sign, one or multiple regime bits, multiple optional exponent bits, and multiple optional fraction bits (Fig. 1). The sign bit is 0 for positive numbers and 1 for negative ones. The number of regime bits is dynamic following a special encoding. After the sign bit , the regime includes a run of 0 or 1, which is terminated by an opposite bit ($\bar{r}$) or at the end of the number format. Similarly, the number of bits for the exponent and fraction is dynamic. A posit number includes the

exponent and fraction only if necessary. Let *m* be the number of identical bits in the regime bits (amber color). If the first bit is zero, the number of zeros (m) represents a negative value (-m). Otherwise, the number of ones minus one (m-1) represents a positive value (m-1). The regime bits realize a scale factor of used, where used = 2^{2es}. Exponent *e* is regarded as an unsigned integer to realize another scale 2^e. Unlike IEEE 754, posit does not use bias for the exponent. Each exponent may be up to a predefined number of bits (*es*). The remaining bits after the regime and the exponent are used for the fraction (f). Similar to IEEE 754, the fraction includes a hidden bit, which is always **1** as posit does not have any DE normal number. Overall, an n-bit posit number (p) can represent the following numbers. Posit number system is first proposed in [1]. It is designed to be used as an alternative to the conventional IEEE floating-point formats [2] in many fields of applications [3], [4], [5]. It has larger dynamic range than IEEE floating-point format. As a result, a small bit-width posit format can meet the numeric requirements of applications while it brings many memory and computation benefit. In addition, its non-uniformed data distribution fits well with the data distribution of some applications, such as deep learning. The 8-bit or 16-bit posit formats are widely used in deep learning systems The reason for usage of Posit multiplier over IEEE 754 standard they are as follows Unique Value Representation**.**

In the posit format, f(a) is always equal to f(b) if a and b are equal, where f is a function. In the IEEE 754, the reciprocals of positive and negative zeros are +∞, −∞, respectively.

Moreover, the negative zero equals positive zero. This implies +∞ = -∞ which is not true. In a floating-point comparison ($a == b$), the result is always false if either *a* or *b* is NaN. This even holds if *a* and *b* has the same bit representation. In posits, however, *a* and *b* are equal if they use the same bit patterns; otherwise, they are not equal. Moreover, the result of an arithmetic operation would be the same over different hardware systems. For instance, in the case of the *Q.W* example at the beginning, posit needs only 24 bits to generate the correct result.

III. LITERATURE REVIEW

1)An approximate and iterative posit multiplier architecture for FPGAs, 2021.

This paper presents the first approximate and iterative posit multiplier architecture. Generally, as more multiplier cores work in parallel at higher frequency, greater speedup is achieved. Therefore, the proposed multiplier design that is smaller and faster than the previous posit multiplier design can be effectively used in applications requiring a wide dynamic range of posit arithmetic. In future work, we will explore whether the approximate and iterative approach can be applied to custom hardware accelerators for a holomorphic encryption scheme supporting approximate arithmetic .

2)A Posit Logarithm-Approximate Multiplier, 2021.

This paper aims to reduce such a gap by proposing a Posit Logarithm-Approximate Multiplication (PLAM) scheme to reduce posit multiplication complexity. The experimental results show that applying PLAM in DNN inference does not affect accuracy. When compared to other posit hardware solutions, the proposed implementation achieves area, power, and delay reduction of 72.86%, 81.79%,

IV. EXISTING WORK

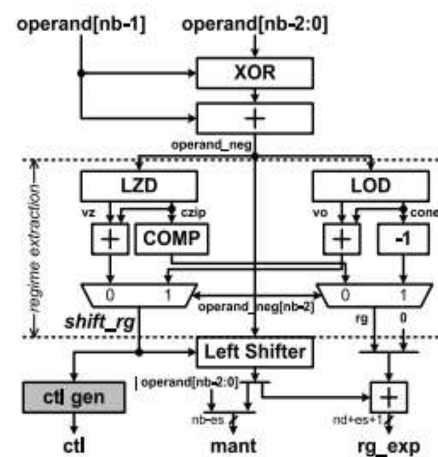

Figure 3.1 Posit Multiplier.

The general format of a posit number is shown in Fig. 1. A posit number Posit (*nb*,*es*) is regime (*rg*), exponent (*exp*), and mantissa (*frac*). The component

bit-width is not constant. The regime bit-width varies for different values. The exponent and the mantissa will occupy the remaining bit positions and they will not be included in the format when the regime occupies all bit positions. The value of a number represented in posits format is:

$$Value = (-1)^s \times useed^{rg} \times 2^{exp} \times (1 + frac)$$

Where *used*= 22^{es}

In hardware arithmetic unit design, the extraction of components is not as straightforward as the floating-point format. The circuit shown in Fig. 3 (except the grey module) is commonly used to extract each component of a posit number [8], [9]. The number is complemented first if it is negative. Then the regime part is first extracted. The regime part is a series of ones (zeros) followed by a single zero (one) bit. Therefore, a leading zero detector (LZD) and a leading one detector (LOD) are used to count the number of leading bits. If leading ones are detected, *rg*equals to *count* - 1. Otherwise, *rg*is *–count* and a complementer, COMP, is needed to convert the positive count to a negative *rg*value. In addition, the regime bit-width *shift_rg*is also generated which is*count*+1 so that the regime can be removed.

1)Modified posit multiplier

The proposed system consists of conventional posit multiplier and modified adder block. A posit number Posit (*nb*, *es*) is defined with the total bit-width *nb*and the exponent bit-width *es*. It has four components: sign (*s*), regime (*rg*), exponent (*exp*), and mantissa (*frac*). The component bit-width is not constant. The regime bit-width varies for different values. The exponent and the mantissa will occupy the remaining bit positions and they will not be included in the format when the regime occupies all bit positions. In parallel adders, carry output of each full adder is given as a carry input to the next higher-order state. Hence, these adders it is not possible to produce carry and sum outputs of any state unless a carry input is available for that state. So, for computation to occur, the circuit has to wait until the carry bit propagated to all states. This induces carry propagation delay in the circuit. The carry input at any stage of the adder is independent of the carry bits generated at the independent stages. Here the output of any stage is dependent only on the bits which are added in the previous stages and the carry input provided at the beginning stage. Hence, the circuit at any stage does not have to wait for the generation of carry-bit from the previous stage and carry bit can be evaluated at any instant of time. A carry-look ahead adder (CLA) or fast adder is a type of electronics adder used in digital logic. A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, ripple-carry adder (RCA), for which the carry bit is calculated alongside the sum bit, and each stage must wait until the previous carry bit has been calculated to begin calculating its own sum bit and carry bit. The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger-value bits of the adder. A LZD circuit is a combinational logic block that determines the number of leading zeros in the primary input word. This LZD block is complex and slow, in general, because its output is a function of all the input bits which can contain from zero -bits (the number bits in the single precision floating-point number significant) to N-bits (the number of bits in the quadruple precision floating-point numbers significant) A N-bit LZD circuit is proposed in this paper. The propagation delay, area and complexity of a LZD block realized with this approach is less than an implementation designed. It waits for the result of the adder to count the number of its leading zeros. A leading zero is any 0 digit that comes before the first nonzero digit in a number string in positional notation. Any zeroes appearing to the left of the first non-zero digit (of any integer or decimal) do not affect its value, and can be omitted (or replaced with blanks) with no loss of information. Therefore, the usual decimal notation of integers does not use leading zeros except for the zero itself, which would be denoted as an empty string otherwise. However, in decimal fractions strictly between −1 and 1, the leading zeros digits between the decimal point and the first nonzero digit are necessary for conveying the magnitude of a number and cannot be omitted, while trailing zeros – zeros occurring after the

decimal point and after the last nonzero digit – can be omitted without changing the meaning. A leading-one detector is an electronic circuit commonly found in central processing units and especially their arithmetic logic units (ALUs). It is used to detect whether the leading bit in a computer word is 1 or 0. Multiplication and addition are the frequently used components in Digital Signal Processing (DSP) applications. Data analysis shows that an average 40% multiplication and 60% addition operations performed in DSP applications. Especially, Finite Impulse Response (FIR), Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT) techniques need to be designed with an efficient multiplier. But, as it is well- known fact that a multiplier has always been a limiting factor in terms of accuracy, speed and area. Design of Leading-One Detector (LOD) is important as they are used for the normalization process in a floating point multiplication, logarithmic multiplication, and in logarithmic converter. The LOD is used in logarithmic converters to find the position of the leading 'one' bit in the integral and the fractional parts a logarithm operation are determined with the help of LOD. An efficient and low power LODs is a demand for logarithmic converter to perform a DSP operation. A LOD is used as a key component for performing the shifting and normalization process in the floating-point multiplication, floating-point addition and also in binary logarithmic converters. Research is going on to evolve various combinatorial circuits in a constrained space with minimum effort. Researchers have continuously working to develop an efficient architecture for LOD The reported LOD design are found to be slower or hardware inefficient. No operations like shifting-and counting method, bit-by-bit serial evaluation circuits etc. make it possible to design the efficient LOD. Further, an effective technique is required to handle the problem of locating of the leading-one bit with a fast, hardware efficient, and low power LOD circuit. It motivates to explore new approaches. Further, the implementation of an efficient architecture of iterative logarithm multiplier is also proposed by using the designed LODs. A comparator compares two input voltages and outputs a binary signal indicating which is larger. If the non-inverting (+) input is greater than the inverting (-) input, the output goes high. If the inverting input is greater than the non-inverting, the output goes low. A shift register is a group of flip-flops, wherein all flip-flops have been inter-connected in such a manner that a binary number can be shifted both inside and outside these flip-flops. In other words, a group of inter-connected flip-flops, on which binary number or binary information can be shifted both inside and outside of these flip-flops, is called shift register. A shift register is also a storage device, wherein binary data or digital information is stored. This device is designed in such a manner that its stored bits can be shifted or transmitted from one flip-flop to another flip-flop (i.e. shift registers are used for storing and shifting of data (0 and 1) in a digital system). Thus, a shift register is a kind of digital circuit, which performs two basic functions i.e. data storing and data shifting. Remember that shift registers are a form of sequential logic circuits, which are being extensively used. The storing capacity of a register or its capacity to store data refers to the number of digital data bits (0 and 1) which it can store or retain inside it. As every stage of a flip-flop present in the shift register reflects storage capacity of a bit (that is every flip-flop of all the flip-flops existing on a register can store just one bit) therefore, the number of stages (flip-flops) in a register represents its overall storing capacity. However, capacity of a register to shift data from one flip-flop to another flip-flop or from one stage to other stage existing within it, or capacity of a register to let data enter into it or let data to be ejected out of it, depends on the use of clock pulses.

In short, a shift register is constructed through binary storage elements (i.e. flip-flops) which are wired or cascaded together in a manner that a bit stored on one element can be shifted to the adjacent element (if flip-flops are connected together in such a way that output of one flip-flop is input for other flip-flop, this process is called cascading of flip-flops). However, it has to be remembered that all storage registers present in a digital system can be made to operate together via an input clock pulse or shift pulse. Therefore, when a shift pulse is applied,

then as per needs, data present on shift register can shift only to one-bit position at a time in a serial manner right or leftward. As a result of this shifting or moving feature just one-bit data at a time, shift registers are widely used for carrying out functions like counting, frequency dividing or arithmetic operations etc. A **binary shift** is a binary operation that consists of shifting all the digit of a binary number either to the left or to the right by a fixed amount. Binary shifts can be used to **multiply a number by a power of 2 (left shift)** or to **divide a number by a power of 2 (right shift).** A **binary left shift** is used to multiply a binary number by two. It consists of shifting all the binary digits to the left by 1 digit and adding an extra digit at the end with a value of 0.A shifter is a circuit that produces an *N*-bit output based on an *N*-bit data input and an *M*-bit control input, where the *N* output bits are place-shifted copies of the input bits, shifted some number of bits to the left or right as determined by the control inputs. As an example, the function of an 8-bit shifter capable of shifting one, two, or three bits to the right or left. The control signals enable several different functions: two bits (A1 and A0) to determine how many bit positions to shift (0, 1, 2, or 3); a fill signal (F) determines whether bits vacated by shift operations receive a '1' or a '0'; a rotate signal (R = '1' for rotate) determines whether shifted-out bits are discarded or recaptured in vacated bits; and a direction signal (D = '1' for right) determines which direction the shift will take. When bits are shifted left or right, some bits "fall off" one end of the shifter, and are simply discarded. New bits must then be shifted in from the opposite side. If no fill input signal exists, then 0's are shifted in (otherwise, the fill input defines whether 1's or 0's are shifted in to vacated bits). Shifters that offer a rotate function recapture shifted-out bits in vacated bits as shown in the lower row. Based on the shifter functions *shift*, *rotate*, *direction*, *fill*, and *number of bits*, many different shifter circuits could be designed to operate on any number of inputs. As an example of a simple shifter design, the truth table in Fig. 2 shows input/output requirements for a four-bit shifter that can shift or rotate an input value left or right by one bit (R=0 for shift, R=1 for rotate, D=0 for left, D=1 for right). Note the truth table uses entered variables to compress the number of rows that would otherwise be required. Shifters are most often found in circuits that work with groups of signals that together represent binary numbers, where they are used to move data bits to new locations on a data bus (i.e., the data bit in position 2 could be moved to position 7 by right-shifting five times), or to perform simple multiplication and division operations (exactly why a bit might want to be moved from one location to another on a data bus is left for a later topic). A shifter circuit can multiply a number by 2, 4, or 8 simply by shifting the number right by 1, 2, or 3 bits (and similarly, a shifter can divide a number by 2, 4, or 8 by shifting the number left by 1, 2, or 3 bits).

2) ADVANTAGES OF PROPOSED SYSTEM

Time consumption consuming under existing method is 14.192ns. The power consumption is 70.427watts. The PDP of exiting posit multiplier 999.499984 JS^{-1} . The proposed output gives enhaced parameters with less power consumption and time delay reduced

V. HARDWARE REQUIREMENTS

1) GENERAL

VLSI stands for "Very Large Scale Integration". This is the field which involves packing more and more logic devices into smaller and smaller areas. VLSI, circuits that would have taken board full of space can now be put into a small space few millimeters across! This has opened up a big opportunity to do things that were not possible before. VLSI circuits are everywhere .your computer, your car, your brand new state-of-the-art digital camera, the cell-phones, and what have you. All this involves a lot of expertise on many fronts within the same field, which we will look at in later sections. VLSI has been around for a long time, but as a side effect of advances in the world of computers, there has been a dramatic proliferation of tools that can be used to design VLSI circuits. Alongside, obeying Moore's law, the capability of an IC has increased exponentially over the years, in terms of computation power, utilization of available area, yield. The combined effect of these two advances is that people can now put diverse

International Journal on Applications in Engineering and Technology
Volume 9: Issue 5 : 2023, pp 16 – 24 www.aetsjournal.com ISSN (Online) : 2455 - 0523

functionality into the IC's, opening up new frontiers. Examples are embedded systems, where intelligent devices are put inside everyday objects, and ubiquitous computing where small computing devices proliferate to such an extent that even the shoes you wear may actually do something useful like monitoring your heartbeats. Integrated circuit (IC) technology is the enabling technology for a whole host of innovative devices and systems that have changed the way we live. Jack Kilby and Robert Noyce received the 2000 Nobel Prize in Physics for their invention of the integrated circuit; without the integrated circuit, neither transistors nor computers would be as important as they are today. VLSI systems are much smaller and consume less power than the discrete components used to build electronic systems before the 1960s. Integration allows us to build systems with many more transistors, allowing much more computing power to be applied to solving a problem. Integrated circuits are also much easier to design and manufacture and are more reliable than discrete systems; that makes it possible to develop special-purpose systems that are more efficient than general-purpose computers for the task at hand.

VI. SOFTWARE REQUIREMENT

VERIFICATION TOOL

- Modalism 6.4c

SYNTHESIS TOOL

- Xilinx ISE 9.1

MODELISM

Modalism is a verification and simulation tool for VHDL, Verilog, System Verilog, and mixed language designs. This lesson provides a brief conceptual overview of the Model Sim simulation environment. It is divided into four topics, which you will learn more about in subsequent lessons.

- Basic simulation flow
- Project flow
- Multiple library flow
- Debugging tools

VII. RESULTS AND DISCUSSION

The proposed and existing designs are modeled in Verilog HDL. These Verilog HDL models are simulated/verified using the Xilinx ISE simulator.

1) Existing posit multiplier

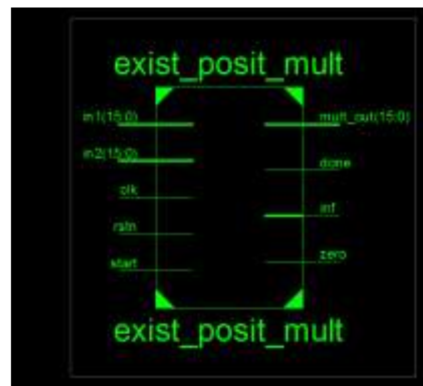

Figure 1 RTL Schematic of existing posit multiplier

Figure 2 RTL Schematic existing posit multiplier

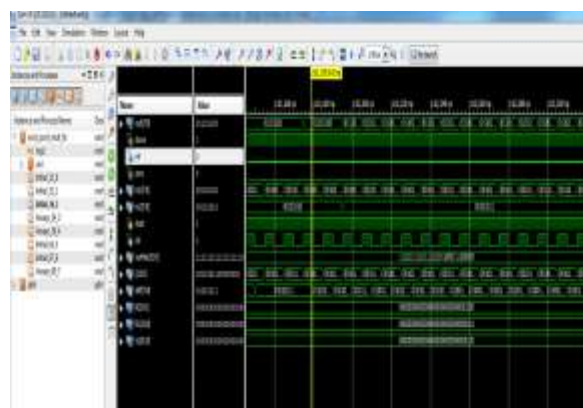

Figure 3 Simulation of existing posit multiplier

2) Proposed posit multiplier

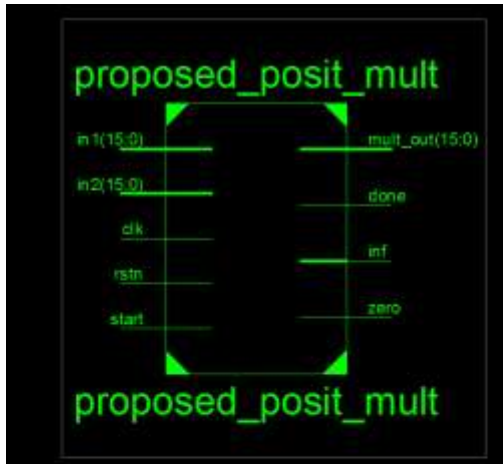

Figure 4 RTL Schematic of existing posit multiplier

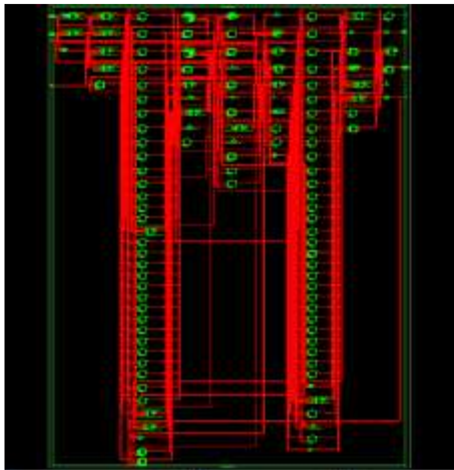

Figure 5 RTL Schematic of existing posit multiplier

VIII. CONCLUSION

The idea proposed in the paper is a 32-bit Posit multiplier architecture with power efficiency. Intrigued by the idea of reconstructing the multiplier unit for mantissa into smaller parts, because of the whole mantissa unit is not used entirely all the time, we have built the hardware. To limit the power consumption, we use only the necessary potion of the multiplier. Our method is evaluated for 16-bit multiplier, whereas we can extend the work for 8-bit and 32-bit posit multipliers using the same technique. For futuristic purposes, more power reduction techniques for multiplier architecture can be developed. The work need not be necessarily limited to multipliers alone. Future works can be deployed also for Posits Adder or Posits Multiply Accumulate functions.

IX. REFERENCES

[1] J. L. Gustafson and I. Yonemoto, “Beating floating point at its own game: IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, Aug. 23, 2008, pp. 1–70.

[2] Z. Carmichael, S. H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, “Deep positron: A deep neural network using the posit number system,” CoRR, vol. abs/1812.01762, pp. 1–6, Dec. 2018.

[3] J. Johnson, “Rethinking floating point for deep learning,” CoRR, vol. abs/1811.01721, pp. 1–8, Nov. 2018.

[4] M. Klöwer, P. D. Düben, and T. N. Palmer, “Posits as an alternative to floats for weather and climate models,” in Proc. Conf. Next Gener. Arithmetic, Mar. 2019, pp. 1–8.

[5] R. Chaurasiya et al., “Parameterized posit arithmetic hardware generator,” in Proc. IEEE 36th Int. Conf. Comput. Design (ICCD), Orlando, FL, USA, Oct. 2018, pp. 334–341.

[6] M. K. Jaiswal and H.-K. So, “Architecture generator for type-3 unum posits adder/subtractor,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Florence, Italy, May 2018, pp. 1–5.

[7] 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), Sapporo, Japan, May 2019, pp. 1–5.

[8] Podobas and S. Matsuoka, “Hardware implementation of POSITs and their application in FPGAs,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW), Vancouver, BC, Canada, May 2018, pp. 138–145.

[9] D. Booth, “A signed binary multiplication technique,” Quart. J. Mech. Appl. Math., vol. 4, no. 2, pp. 236–240, 1951.

[10] 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, Mar. 2019, pp. 1–9

[11] Uguen, Yohann, Luc Forget, and Florent de Dinechin. "Evaluating the hardware cost of the posit number system." 2019 29th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2019.

[12] Murillo, R., Del Barrio, A. A., Botella, G., Kim, M. S., Kim, H., &Bagherzadeh, N. (2021). PLAM: A posits logarithm-approximate multiplier. IEEE Transactions on Emerging Topics in Computing, 10(4), 2079-2085.